DTIC FILE COPY

(2)

DOT/FAA/CT-86/27

FAA Technical Center
Atlantic City Airport,
New Jersey 08405

SOFTWARE DEPENDABILITY ASSESSMENT METHODS

Shirley A. Prater
Ellis F. Hitt
Donald Eldredge

BATTELLE
Columbus Division
505 King Avenue
Columbus, Ohio 43201

November 1986

Final Report

This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

U.S. Department of Transportation
Federal Aviation Administration

1. Report No. DOT/FAA/CT-86/27	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle SOFTWARE DEPENDABILITY ASSESSMENT METHODS		5. Report Date November 1986
		6. Performing Organization Code
7. Author(s) Shirley A. Prater, Ellis F. Hitt, & Donald Eldredge		8. Performing Organization Report No. DOT/FAA/CT-86/27
9. Performing Organization Name and Address Battelle Columbus Division 505 King Avenue Columbus, Ohio 43201		10. Work Unit No.
		11. Contract or Grant No. NAS2-11853
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Technical Center Atlantic City Airport, NJ 08405		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes
Point of Contact: W. E. Larsen/MS:210-2
Ames Research Center
Moffett Field, CA 94035

16. Abstract

The purpose of this document is to identify various software reliability models, define the interface between a software reliability model with a fault tolerant system reliability model, and provide a software dependability model (capable of evaluating availability, reliability, and safety) that can predict the reliability of software prior to and throughout its development. The software reliability data and development of a software reliability data base is also discussed.

17. Key Words (Suggested by Author(s)) Software reliability, safety, availability, dependability, fault tolerance, N-version software, recovery block, software reliability data base	18. Distribution Statement Unlimited Subject Category 38

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages	22. Price*

*For sale by the National Technical Information Service, Springfield, Virginia 22161

A

PREFACE

The dependability property of a computer system allows reliance to be justifiably placed on the service it delivers, which is its behavior as perceived by its users [AVLA86]. This concept naturally encompasses the notions of reliability, availability, and safety.

The purpose of dependability is the design, implementation, and use of computer systems where faults are natural, foreseeable, and tolerable [TOUL76].

Dr. John P. J. Kelly

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES (Continued)

TECHNICAL REPORT

on

SOFTWARE DEPENDABILITY ASSESSMENT METHODS

1.0 INTRODUCTION

The increasing application and criticality of digitally implemented flight control functions dictates a need for highly dependable software. A variety of methods for creating reliable software have been developed (e.g., software engineering, higher order languages, testing and debugging procedures). These methods do not inherently provide a measure of reliability improvement obtained using the methods. Methods of assessing the reliability and dependability of software are needed. This effort was originally entitled "software reliability assessment methods" but as the work progressed, Battelle and our consultant, Dr. John P. J. Kelly, determined that systems which provide functions critical to the safe transportation of passengers must be more than reliable, they must also be dependable. As stated in the preface written by Dr. Kelly, the concept of dependability encompasses the notions of reliability, availability, and safety.

The overall objective of this research was to investigate methods of assessing the reliability of digtial avionics software developed using primarily higher order languages.

2.0 BACKGROUND

Many software reliability models have been developed which predict the reliability of software based on various input parameters. That work was sponsored by the USAF Rome Air Development Center (RADC) and NASA's Ames and Langley Research Centers. Many of these studies attempted to analyze an existing data base to derive a prediction methodology. Many of these data bases were files of historical significance, and were not real-time airborne software systems developed using higher order languages. Consequently, many researchers have found the popular software reliability models are invalid. Recent work involved carfully designed and controlled software development experiments using a limited number of programmers programming a limited number of problems. That work, in turn, has been criticized as not representative of real-time digital avionics software.

3.0 SUMMARY

This report is in five sections. The following four sections present:

(1) a summary of the results of previous studies of software reliability models applicable to real-time software

(2) a definition of the software dependability model interfaces with fault tolerant system reliability models

(3) formulation of a software dependability model

(4) a definition of software data to be collected by the avionics system developer for storage in the software dependability data base.

TECHNICAL REPORT

on

REVIEW OF PREVIOUS STUDIES OF SOFTWARE RELIABILITY MODELS

1.0 INTRODUCTION

Several studies of software reliability models have been reviewed. In addition to those models covered in the "Comparative Analysis of Fault Tolerant Software Design Techniques", prepared by Battelle Columbus Division on February 15, 1984, other models have been reviewed through the use of published technical papers and a software engineering textbook.(1) The technical papers are listed in Table 1 and divided into the following categories: "For Information Only", "Discusses Reliability Modeling Applicable to Real-Time Software", and "Discusses Reliability Modeling Not Applicable to Real-Time Software". Table 2 gives the names of the reliability models and indicates whether or not they are applicable to real-time software. Only the software reliability models which are applicable to real-time software will be discussed in further detail.

The following software reliability models have the characteristic of being applicable to real-time software. The criteria used to determine real-time applicability was:

(a) The model is not extremely difficult to solve numerically. Hence, only a reasonable amount of computations is required.

(b) The model must take into account the test time.

(1) Martin L. Shooman
SOFTWARE ENGINEERING Design/Reliability/Management,
McGraw-Hill Book Company, New York, 1983.

Title	Author(s)	Bibliographical Information	For Information Only	Discusses Reliability Modeling Applicable to Real-Time Software	Discusses Reliability Modeling Not Applicable to Real-Time Software
Software Reliability: A Historical Perspec--tive	Martin L. Shooman	*IEEE Transactions On Reliability*, Volume R-33, Number 1, April 1984, pp. 48-55.	x		
Criteria for Software Reliability Model Comparisons	Anthony Iannino, John D. Musa, Kazuhira Okumoto, and Bev Littlewood	*IEEE Transactions On Software Engineering*, Volume SE-10, Number 6, November 1984, pp. 687-691.	x		
How to Measure Software Reliability and How Not To	Bev Littlewood	*IEEE Transactions On Reliability*, Volume R-28, Number 2, June 1979, pp. 103-110.	x		
Theories of Software Reliability: How Good Are They and How Can They Be Improved?	Bev Littlewood	*IEEE Transactions On Software Engineering*, Volume SE-6, Number 6, September 1980, pp. 489-500.	x		
Models for Failure	R. A. Evans	*IEEE Transactions On Reliability*, Volume R-29, Number 5, December 1980, p. 353.	x		
A Conditional Probability Approach to Reliability with Common-Cause Failure	John Yuan	*IEEE Transactions On Reliability*, Volume R-34, Number 1, April 1985, pp. 38-42.			x
The Linear Software Reliability Model and Uniform Testing	Martin Trachtenberg	*IEEE Transactions On Reliability*, Volume R-34, Number 1, April 1985, pp. 8-16.		x	x
Bayesian Reliability and Availability	Frank A. Tillman, Way Kuo, C.L. Hwang, and Doris Lloyd Grosh	*IEEE Transactions On Reliability*, Volume R-31, Number 4, October 1982, pp. 362-372.			x
The Conservativeness of Reliability Estimates Based on Instantaneous Coverage	John McGough, Mark Smotherman, and Kishor S. Trivedi	*IEEE Transactions On Computers*, Volume C-34, Number 7, July 1985, pp. 602-609.		x	x
A Summary of the Discussion on "An Analysis of Competing Software Reliability Models"	Bev Littlewood	*IEEE Transactions On Software Engineering*, Volume SE-6, Number 6, September 1980, pp. 501-502.		x	x
A Theory of Software Reliability and Its Application	John D. Musa	*IEEE Transactions On Software Engineering*, Volume SE-1, Number 3, September 1975, pp. 312-327.			x
Stochastic Reliability-Growth: A Model for Fault Removal in Computer Programs and Hardware Designs	Bev Littlewood	*IEEE Transactions On Reliability*, Volume R-30, Number 4, October 1981, pp. 313-320.		x	x
Software Reliability as an Application of Martingale & Filtering Theory	G. Koch and P.J.C. Spreij	*IEEE Transactions On Reliability*, Volume R-32, Number 4, October 1983, pp. 342-345.		x	x
Discrete Probability Models with Modified Zeros	J.J. Higgins and C.P. Tsokos	*IEEE Transactions On Reliability*, Volume R-27, Number 5, December 1978, pp. 363-366.		x	x
S-Shaped Reliability Growth Modeling for Software Error Detection	Shigeru Yamada, Mitsuru Ohba, and Shunji Osaki	*IEEE Transactions On Reliability*, Volume R-32, Number 5, December 1983, pp. 475-478.		x	x
Birth-Death and Bug Counting	Wilhelm Kremer	*IEEE Transactions On Reliability*, Volume R-32, Number 1, April 1983, pp. 37-47.		x	x
Estimating the Number of Faults in Code	John E. Gaffney, Jr.	*IEEE Transactions On Software Engineering*, Volume SE-10, Number 4, July 1984, pp. 459-464.	x		

TABLE 1. BREAKDOWN OF TECHNICAL PAPERS REVIEWED

Software Reliability Model	Applicable to Real-Time Software	Not Applicable to Real-Time Software					
		Reasons*					
		a	b	c	d	e	f
Ramamoorthy and Bastani's Input Domain Based Model					x	x	
Nelson's Input Domain Based Model					x		
Ho's Input Domain Based Model		x			x		
Error Seeding Model by Mills				x			x
Experimental Reliability Data Model		x		x	x		
Path Decomposition Model by Shooman					x		
Imperfect Debugging Model by Goel					x		x
Generalized Imperfect Debugging Model	x						
Stochastic Model					x		
Bug - Proportional Model	x						
Pragmatic Software Reliability Prediction Model by Wall and Ferguson						x	
Generalized Poisson Model by Weibull (Wagoner)					x		
Geometric Poisson Model by Moranda	x						
Goel-Okumoto Non-Homogeneous Poisson Model					x		
Schneidewind Non-Homogeneous Poisson Model	x						
Goel Modified Non-Homogeneous Poisson Model					x		
Green Model		x					
Weibull (Coutinho) Model				x			
Corcoran-Weingarten-Zehna Model			x		x		
Weiss Model							x
Musa Execution Time Failure Model		x				x	x
Jelinski-Moranda De-Eutrophication Model	x						
Extended Jelinski-Moranda Model	x						
Geometric De-Eutrophication Model by Moranda	x						
Modified Geometric De-Eutrophication Model	x						
LaPadula Reliability Growth Model			x		x		

TABLE 2. REVIEWED SOFTWARE RELIABILITY MODELS

Software Reliability Model	Applicable to Real-Time Software	Not Applicable to Real-Time Software					
		Reasons*					
		a	b	c	d	e	f
Schick-Wolverton Linear Model					x		x
Extended Schick-Wolverton Model by Lipow					x		x
Modified Schick-Wolverton Model by Sukert					x		x
Shooman Structural Model		x				x	
Shooman Exponential Model	x						
Shooman-Natarajan Manpower Limited Model					x		x
Miyamoto's Revised Shooman Model							x
Markov Model by Shooman and Trivedi					x		
Littlewood Decreasing Failure Rate Model (Bayesian Debugging Model)						x	
Littlewood Markov Model		x					
Bayesian Reliability Growth Model by Littlewood and Verrall				x		x	x

* Reasons:

a. The model is extremely difficult to solve numerically. Consequently, the variables must be selected so as to limit the amount of computation required.

b. The test time is totally ignored.

c. The technique(s) used is (are) undesirable in software reliability models.

d. The model does not adequately address the overall software reliability problem.

e. There are massive difficulties in estimating the parameters.

f. Some of the assumptions are quite questionable.

TABLE 2. REVIEWED SOFTWARE RELIABILITY MODELS (Continued)

(c) The techniques used in the model must be reasonable. (For example, the cumulative averaging technique is undesirable in software reliability models because this technique causes early data points to be weighted more heavily than later ones. Thus, with this technique, even the most erratic error data will eventually project a "fitted line" as the sample size becomes sufficiently large).

(d) The model addresses both the dynamic and static measurements:

(1) Dynamic measurements
- hazard rate, $z(t)$
- reliability function, $R(t)$
- mean time to failure, MTTF

(2) Static measurements
- total number of errors
- total number of remaining errors

(e) The parameters are not diffcult to estimate.

(f) The assumptions are reasonable.

The underlying assumptions, key features, and study results are summarized below.

2.0 GENERALIZED IMPERFECT DEBUGGING MODEL

2.1 Underlying Assumptions

a. All failures are observable and independent.

b. The time to remove a failure is considered to be negligible and is ignored in the model.

c. Errors are not always corrected when detected and errors may be spawned when correcting errors.

d. Testing is of uniform intensity and representative of the operational environment.

e. Inputs which exercise the program are randomly selected.

f. The failure rate at any time is proportional to the current number of errors remaining in the program.

g. The failure rate between the (i-1)th failure and the ith failure is $\lambda(t_i) = \phi[N-(i-1)]t^{\alpha-1}$.

2.2 Key Features

The failure rate is of the form

$$\lambda(t) = \phi(N-p(i-1))t^{\alpha-1}$$

with ϕ = a proportionality constant;

N = the total number of errors;

p = the probability of perfect programmer debugging behavior

α = the parameter that controls the shape of the failure rate.

The reliability functions is

$$R(t) = \exp\left(-\frac{\phi\,(N-p(i-1))}{\alpha}\,t^{\alpha}\right)$$

and the Mean Time to Failure is

$$MTTF = \frac{1}{\alpha}\left\{\frac{\alpha}{(N-p(i-1))}\right\}^{1/\alpha}\Gamma\left(\frac{1}{\alpha}\right).$$

2.3 Study Results

This model cannot determine the effect each bug contributes to the overall failure rate without continuing to run the program because the instantaneous failure rate will be zero when a bug is found and immediately removed. The model provides a good fit with data. The parameter estimates are reasonable for the data sets tested.

3.0 BUG-PROPORTIONAL MODEL

3.1 Underlying Assumptions

a. The number of errors in a program is a constant and decreases directly as errors are corrected.

b. Software errors are caused by the uncovering of residual bugs in a program.

c. The probability that a bug is encountered in the time interval, Δt, after t successful hours of operation is proportional to the fractional number of remaining bugs.

d. The fractional number of remaining bugs is independent of the operating time.

e. The rate of error correction is constant.

3.2 Key Features

The hazard rate is of the form

$$z(t) = K\epsilon_r(\tau) = K[(E_T/I_T) - \epsilon_c(\tau)]$$

where K = an arbitrary constant; and

$\epsilon_r(\tau)$ = the number of remaining bugs;

E_T = the total number of errors originally present;

I_T = the total number of machine instructions; and

$\epsilon_c(\tau)$ = the number of corrected bugs.

The reliability function is

$$R(t) = \exp\{-[K\epsilon_r(\tau)]t\} = \exp\{-K[(E_T/I_T) - \epsilon_c(\tau)]t\}$$

and the Mean Time to Failure is

$$\text{MTTF} = \frac{1}{K\epsilon_r(\tau)} = \frac{1}{\beta(1-\alpha\tau)} \text{ with } \beta = \frac{E_T}{I_T}K \text{ and } \alpha = \frac{\rho_0 I_T}{E_T},$$

where ρ_0 = a constant rate of error correction.

3.3 Study Results

The overall behavior of the model is verified. However, the errors between measurement and prediction had a standard deviation of 24 percent. When seeking historical data for estimation of reliability parameters, the examples should closely match the intended application and phase.

4.0 GEOMETRIC POISSON MODEL

4.1 Underlying Assumptions

a. There is an infinite number of errors.

b. Each fault in the program is independent of the others and each of them is equally likely to occur.

c. The errors do not have the same likelihood of detection.

d. During a fixed interval of time, the number of errors detected follows a Poisson distribution.

e. During each of these periods of time, the detection rate is constant.

f. Data is available only at discrete intervals.

g. The detection rate in successive time intervals forms a geometric progression.

h. Each error discovered is immediately removed or no longer counted.

i. No new fault is introduced during a correction time.

4.2 Key Features

The hazard rate during the ith time interval is

$$z(t_i) = \lambda K^{i-1}$$

where t_i = the ith debugging interval;

λ = the average number of faults occurring in the first interval;

K = a proportionality constant, $0 < K < 1$.

The reliability function is

$$R(t) = e^{-\lambda K^i t}$$

and the Mean Time to Failure is

$$MTTF = \frac{1}{\lambda K^i} .$$

4.3 Study Results

This model gives identical results as the Schneidewind Non-Homogeneous model.

5.0 SCHNEIDEWIND NON-HOMOGENEOUS POISSON MODEL

5.1 Underlying Assumptions

a. The number of errors which is detected during a time interval and the collection of error counts over a series of time intervals are modelled by a random variable and a stochastic process.

b. Prior to the selection of a test plan, all errors are equally likely.

c. The number of errors detected in each time interval is independent of the number detected in another time interval.

d. Detected error counts in each interval have the same type of distribution but have different means.

e. The mean number of detected errors decreases from interval to interval.

f. The rate of detection in an interval is proportional to the number of errors in that interval.

g. The error process is a non-homogeneous Poisson process with an exponentially decreasing intensity function.

h. The error correction rate is proportional to the number of errors to be corrected.

5.2 Key Features

The hazard function is equivalent to the predicted number of errors for each interval i where

$$m_i = (\alpha/\beta)[\exp(-\beta(i-1)) - \exp(-\beta i)]$$

with m_i = the estimated number of errors in interval i;

β = a model constant; and

α = a model constant (error detection rate at time 0).

The weighted squared deviation is

$$SD_W = \sum_{k=1}^{t} \exp(\beta i)[(\alpha/\beta)[\exp(-\beta i)][\exp(\beta)-1] - X_i]^2.$$

5.3 Study Results

This model is equivalent to the Geometric De-Eutrophication model. However, this model offers greater flexibility than the Geometric De-Eutrophication model.

The required test data for this model consists of the sequence of the number of errors in each time interval.

6.0 JELINSKI-MORANDA DE-EUTROPHICATION MODEL

6.1 Underlying Assumptions

a. A program can be decomposed into a number of paths or cases.

b. The identification of paths will be done at a high enough level to yield a relatively small number of cases ($\leq 10^{20}$).

c. The number of machine language instructions remains relatively constant.

d. Failure is caused by rare combinations of input data and path traversals, with the time between failures governed by an exponential distribution, yielding a constant hazard.

e. There is a fixed number of errors in the program.

f. No new errors are added during the debugging process.

g. Each error discovered is immediately removed.

h. Each error has an equal chance of being detected.

i. The failure rate is proportional to the current error content (number of remaining errors).

j. The program is not being altered except for error correction.

k. Only one error may occur in a given time debugging period.

6.2 Key Features

The hazard function is of the form

$$z(X_i) = \phi[N-(i-1)]$$

with N = the total number of initial errors in the program;

ϕ = a proportionality constant;

X_i = the length of the ith debugging interval (the time between detection of the (i-1)st and the ith errors); and

i = the number of errors discovered.

The reliability function is

$$R(X_i) = \exp[-\phi(N-n)X_i]$$

and the Mean Time to Failure is

$$MTTF = 1/[\phi(N-n)]$$

where n = the number of errors found to date.

6.3 Study Results

The model provides a good fit with data. The model runs into slight trouble with its "no new errors" assumption. At the "last" error, the time between errors shows a sudden sharp increase which is somewhat optimistic. The larger the data set, the more likely the sudden improvement and the implication that debugging is complete.

This model requires a sequence of times between failures in order to estimate the parameters.

7.0 EXTENDED JELINSKI-MORANDA MODEL

7.1 Underlying Assumptions

a. There is a fixed number of errors in the program.

b. No new errors are added during the debugging process.

c. Each error discovered is immediately removed.

d. Each error has an equal chance of being detected.

e. There is a constant failure rate between consecutive errors.

f. A program can be decomposed into a number of paths or cases.

g. The identification of paths will be done at a high enough level to yield a relatively small number of cases ($\leq 10^{20}$).

h. The number of machine language instructions remains relatively constant.

i. The failure rate is proportional to the current error content (number of remaining errors).

j. The program is not being altered except for error correction.

k. More than one error may occur in a given time debugging period.

7.2 Key Features

The hazard function is of the form

$$z(t_i) = \phi[N-n_{i-1}]$$

where ϕ = a proportionality constant;

N = the total number of initial errors;

n_i = the cumulative number of errors found through the ith time interval; and

t_i = the ith debugging interval.

The reliability function is

$$R(t) = e^{-\phi(N-n_i)t}$$

and the Mean Time to Failure is

$$MTTF = \frac{1}{\phi[N-n_i]}.$$

7.3 Study Results

This model requires the number of errors in some uniform time period to estimate the parameters.

The model provides a good fit with data. The model is somewhat inconsistent and less smooth due to its use of actual errors in its hazard function. Fairly small changes in the data give a significant change in the model's shape and prediction. At the "last" error, the time between errors shows a sudden sharp increase which is somewhat optimistic.

8.0 GEOMETRIC DE-EUTROPHICATION MODEL

8.1 Underlying Assumptions

a. There is an infinite number of total errors.

b. Each fault in the program is independent of the others and each of them is equally likely to occur.

c. The errors do not have the same likelihood of detection.

d. Each error discovered is immediately removed. The time to correct the detected faults is negligible.

e. No new fault is introduced during the correction time.

f. The failure rate between successive errors forms a geometric progression and is constant in the interval between errors.

8.2 Key Features

The hazard function is

$$Z(X_i) = DK^{i-1}$$

where X_i = the ith debugging interval;
D = the initial error detection rate;
K = a proportionality constant; and
i = the number of errors discovered after i intervals.

The reliability function is

$$R(X_i) = \exp[-DK^n X_i]$$

where n = the total number of errors discovered

and the Mean Time to Failure is

$$MTTF = \frac{1}{DK^n}.$$

8.3 Study Results

The test data necessary to apply this model is the sequence of times between errors.

This model gives a reasonable fit with data. It is also fairly consistent in its results when only part of the data is used. The Geometric De-Eutrophication model appears to be slightly better than the Jelinski-Moranda De-Eutrophication model for data.

9.0 MODIFIED GEOMETRIC DE-EUTROPHICATION MODEL

9.1 Underlying Assumptions

a. The program contains an unknown number of errors.

b. Each fault in the program is independent of other faults and each of them is equally likely to cause a failure during testing.

c. The number of faults detected in any time interval is independent of that in any other time interval.

d. The error correction time is negligible. Each error discovered is immediately removed.

e. No new errors are added during the debugging process.

f. The program is not being altered except for error correction.

9.2 Key Features

The hazard function is of the form

$$z(t_i) = DK^{M_{i-1}}$$

with D = the fault detection rate;
K = a positive constant less than 1;
M_{i-1} = the cumulative number of errors detected; and
M_i = the cumulative number of errors found up to the i-th time interval.

The reliability function is

$$R(t) = e^{-DK^{M_n}t}$$

and the Mean Time to Failure is

$$MTTF = \frac{1}{DK^{M_n}},$$

with n = the total number of time intervals.

9.3 Study Results

This model was not verified with any test data.

10.0 SHOOMAN EXPONENTIAL MODEL

10.1 Underlying Assumptions

a. The number of errors in a program is a constant and decreases directly as errors are corrected.

b. The error detection rate (failure rate) is proportional to the number of remaining errors.

c. The total number of machine language instructions remains constant.

d. Operational software errors occur due to the occasional traversing of a portion of the program in which a software bug is hidden.

e. Each error has an equal chance of being detected.

f. Software errors occur with a probability distribution of

$$f(t) = \lambda \exp(-\lambda t)$$

where t = CPU operating time; and
λ = a constant of the hazard function, $z(t)$.

10.2 Key Features

The total number of errors remaining in the program debug time τ is

$$e_r(\tau) = (E/I) - e_c(\tau)$$

where E = the total number of errors present at time $\tau = 0$;
I = the total number of machine instructions; and
$e_c(\tau)$ = the total number of errors corrected in interval τ.

The hazard function is of the form

$$z(t) = Ce_r(\tau)$$

with C = a constant of proportionality.

The reliability function is

$$R(t) = \exp\{-C[(E/I) - e_c(\tau)]t\}$$

and the Mean Time to Failure is

$$MTTF = 1/\{C[(E/I) - e_c(\tau)]\}.$$

10.3 Study Results

The Shooman exponential model reduces to the Jelinski-Moranda De-Eutrophication model. This model requires a sequence of times between failures in order to estimate the parameters.

11.0 BIBLIOGRAPHY

1. Chen, Yao, "Time Dependent Software Reliability Modeling Study", North Carolina State University, Raleigh, North Carolina, 1982.

2. Gephart, L. S., Greenwald, C. M., Hoffman, M. M., and Osterfeld, D. H., "Software Reliability: Determination and Prediction", University of Dayton, Dayton, Ohio, Report Number AFFDL-TR-78-77, June 1978.

3. Goel, Amrit L., "A Guidebook for Software Reliability Assessment", Syracuse University, Syracuse, New York, Report Numbers RADC-TR-83-16 and AD-A 39240, August 1983.

4. Hitt, Ellis F., Webb, Jeffrey J., and Bridgman, Michael S., "Comparative Analysis of Fault Tolerant Software Design Techniques", Battelle Memorial Institute, Columbus, Ohio (Prepared Under Contract Number NAS1-17412), February 15, 1984.

5. Littlewood, Bev and Sofer, Ariela, "A Bayesian Modification to the Jelinski-Moranda Software Reliability Growth Model", City University, London, England, Report Number NASA-OR-169743, 1983.

6. Shooman, Martin L., <u>SOFTWARE ENGINEERING Design/Reliability/ Management</u>, McGraw-Hill Book Company, New York, 1983.

7. Sukert, Alan N., "A Software Reliability Modeling Study", Rome Air Development Center, Griffiss Air Force Base, New York, Report Numbers RADC-TR-76-247 and AD/A-030-437, August 1976.

TECHNICAL REPORT

on

DEFINTION OF THE FAULT TOLERANT SYSTEM RELIABILITY MODEL INTERFACES WITH THE SOFTWARE RELIABILITY MODEL

1.0 INTRODUCTION

1.1 Background

Interface checks are an attractive form of error detection (at least as far as a program running on the interface is concerned) since they are performed automatically and efficiently - often in parallel with the execution of the requested operation - and cannot be suppressed by a programmer. However, interface checks can only check for correctness of use of an interface and cannot check whether any usage corresponds to that of a correct program. It has been assumed that interface exceptions and failure exceptions indicated the presence of design faults in the program and component faults in the interpreter, respectively. If this was always the case then the implementation of fault tolerance by the program would be much simplified since there would be a direct relationship between the type of a fault and a particular exception.

1.2 Objectives of the Research

To define the interface of the software reliability model with the fault tolerant system reliability model, it is necessary to look at the corresponding outputs and required inputs. It is desired that the interface definition permit stand alone use of the software reliability model with the outputs serving as inputs to the fault tolerant system reliability model or a combined operation of the hardware and software models of the state space. The outputs and inputs are discussed in detail below.

2.0 CHARACTERISTICS OF THE SOFTWARE RELIABILITY MODEL AND THE FAULT TOLERANT SYSTEM RELIABILITY MODEL

2.1 Software Reliability Model Outputs

Software reliability models that are applicable to real-time software address both the dynamic and static measurements. The corresponding outputs are:

(1) Dynamic measurements
- hazard rate, $z(t)$
- reliability function, $R(t)$
- mean time to failure, MTTF

(2) Static measurements
- total number of errors
- total number of remaining errors

2.2 Fault Tolerant System Reliability Models

In accordance with the "Automated Reliability and Failure Effects Method for Digital Flight Control and Avionic Systems, Volume I: Evaluation", this report will focus on the top two models. The evaluation lists the top two models, in order of preference, as CARSRA (Computer Aided Redundant System Reliability Analysis) and CARE II (Computer Aided Reliability Estimation, version II). A different evaluation, sponsored by NASA Langley Research Center, indicates that the CARE III model is "best suited for evaluating the reliability of advanced fault-tolerant systems for commercial air transport."(1) Therefore, this report will discuss the updated and improved CARE model: CARE III. In addition, N-Version Software (NVS), obtained from multi-version programming, will be considered since it is a primary method for providing software fault tolerance and was not covered in the above reports.

(1) "Evaluation of Reliability Modeling Tools for Advanced Fault-Tolerant Systems", AIRLAB INTERFACE, NASA Langley Research Center, Hampton, Virginia, December 1983, p. 2.

2.2.1 CARSRA Inputs

The following items comprise the required input data and the corresponding variable names that are used by the program.

- number of non-dependency stages, NIS
- number of dependency stages, NDS
- assigned state number, NST
- dimension of the stage, NDIM
- number of modules in the stage, MODN
- transition rates, LMDA (NST, K, J) with J=1, 2,...NDIM and K=1, 2,...(NDIM-1)
- transitional readiness time span, AMT, and time increment, ADT
- failure probability time span, FPMT, and time increment, FPDT
- number of dependency modules in the system, NARY
- each dependency module, NIND (I) with I=1, 2,...NARY, specified by NIND and NDEP
- number of functional readiness configuration entries, NAV
- each configuration is characterized by up to three failed modules, NA (I, K), with K=1, 2, 3 where the module is indicated by XXY with XX being the stage number and Y the module number within the stage
- number of stage failure patterns equivalent to system success, NOSCOF
- success configurations, ICOF (I, J) with I=NOSCOF and J=1, 2,...50
- accuracy indicator, NACCUR

2.2.2 CARE III Inputs

The following input data is required to describe the system. The corresponding variable names are given after the description.

- variable which defines if all of the fault handling models have exponential distributions only, MARKOV
- number of fault types to be included in the model, NFTYPS
- parameter for transition between the active state (A or A_E) to the detected state (D), DEL(i)
- parameter for transition from the active state A to the active error (erroneous operation state) A_E, RHO(i)
- parameter for transition from the error producing state to the detected state or to a single fault failure, EPS(i)
- indicator variable defining if the DEL parameter is for an exponential or uniform density, IDELF(i) - disregard if it is a Markovian model
- indicator variable defining if the RHO parameter is for an exponential or uniform density, IRHOF(i) - disregard if it is a Markovian model
- indicator variable defining if the EPS parameter is for an exponential or uniform density, IEPSF(i) - disregard if it is a Markovian model
- probability that a faulty operation will be successfully masked by the system, C(i)
- probability that a module detected as faulty in an active state A is identified as a permanent fault and isolated from the system, PA(i)
- probability that a module detected as faulty in a benign state is identified as a permanent fault and isolated from the system (for intermittent or transient fault), PB(i)
- exponential rate (intermittent or transient fault) for transition from an active state (A or A_E) to a benign state (B or B_E), ALP(i)
- exponential rate (intermittent fault) for transition from a benign state (B or B_E) to an active state (A or A_E), BET(i)

- flag for outputting the moments of single and double fault coverage functions, CVPRNT
- flag for a plot of the single and double fault coverage functions, CVPLOT
- Y-axis scale for plotting coverage functions, IAXSCV
- parameter governing the step doubling rule used in the solution, DBLDF
- coverage function's truncation value, TRUNC
- number of stages in the system, NSTGES
- number of identical modules in stage number x, N(x)
- minimum number of modules needed for stage ISTG to be operational, M(x)
- operational configurations for stage x, NOP (i,x)
- option for the output printout, IRLPCD
- option for the summary information to be plotted against time, RLPLOT
- axes specification for the summary information plot, IAXSRL
- number of fault types that a stage is subject to, NFCATS(x)
- fault types specification for stage x, JTYP(j,x)
- parameter ω of the Weibull fault occurrence rate $\lambda\omega(\lambda t)^{\omega-1}$ for fault type j for stage x, OMG(j, x)
- parameter λ of the Weibull fault occurrence rate for the fault type j for stage x, RLM(j, x)
- flight time for which the system is to be assessed, FT
- time scale used for the flight time, ITBASE
- number of equal steps that the flight time is divided into, NSTEPS
- flag indicating whether or not the system fault tree is to follow, SYSFLG
- flag indicating whether or not a critical pairs fault tree is provided, CPLFLG

- parameter used to limit the number of terms used in computing the coverage failure probability, PSTRNC
- parameter used to limit the number of fault vectors used in computing the probability of system failure due to a lack of coverage, QPTRNC
- parameter affecting the computation of the summary information, KWT
- identification label for the system represented, TITLE
- logic statements to form a single system fault tree
- identification label for the critical pair tree, TITLE
- logic statements for the critical pair tree

2.2.3 NVS Inputs

The following parameters are necessary for the NVS reliability analysis.

- number of active versions, n
- error probability of version #i (with $q_i=1-p_i$), p_i
- correlation coefficient between #i and #j (with $q_{i,j}=1-p_{i,j}$), $p_{i,j}$
- maximum number of versions allowed to be faulty at any crosscheck point, m*
- maximum number of versions allowed to be faulty in common mode, f*

3.0 INTERFACE DEFINITION

It is desired that the software reliability model be able to be used in a stand alone environment or in conjunction with the fault tolerant system reliability model. Therefore, the interface definition shall dictate which outputs of the software reliability model must serve as inputs to the fault tolerant system reliability model. For clarification, this will be done individually for the three models: CARSRA, CARE III, and NVS.

3.1 Interface Definition with CARSRA

The required inputs for the CARSRA model are from three different sources: (1) inputs into the software reliability model; (2) outputs from the software reliability model; and (3) inputs only for the fault tolerant system reliability model. Table 1 shows the distribution of the various CARSRA inputs between these three categories. The inputs from the first two categories are transferred from the software reliability model into the fault tolerant system reliability model at the interface. Hence, the interface definition for the CARSRA model is shown in Figure 1. Table 2 shows the relationship between the various outputs from the software reliability model and the CARSRA inputs (given in the second column in Table 1).

3.2 Interface Definition with CARE III

Similar to the CARSRA model, the required inputs for the CARE III model are from the following three sources: (1) inputs into the software reliability model; (2) outputs from the software reliability model; and (3) inputs only for the fault tolerant system reliability model. The distribution of the CARE III inputs amongst these three categories is shown in Table 3. Figure 1 remains applicable for defining the interface of the software reliability model with the fault tolerant system reliability model. The correspondence between the software reliability model output variables and the CARE III input variables (given in the second column in Table 3) is shown in Table 4.

3.3 Interface Definition with NVS

For the NVS model, the required inputs are in the following two categories: (1) outputs from the software reliability model and (2) inputs only for the fault tolerant system reliability model. Table 5 shows the breakdown of the NVS inputs into these categories and Table 6 gives the relationship of the NVS inputs to the software reliability model outputs.

Input into the Software Reliability Model	Output from the Software Reliability Model	Input only into the Fault Tolerant System Reliability Model
• number of non-dependency stages, NIS • number of dependency stages, NDS • assigned stage number, NST • dimension of the stage, NDIM • number of modules in the stage, MODN	• transition rate matrix in failures per million hours, LMDA (NST, K, J) with J=1, 2,...NDIM and K=1, 2,...(NDIM-1) • transitional readiness time span, AMT, and time increment, ADT • failure probability time span, FPMT, and time increment, FPDT • number of failed modules characterizing each configuration, NA (I, K) • number of stage failure patterns equivalent to system success, NOSCOF • success configurations, ICOF (I, J) • accuracy indicator, NACCUR	• number of dependency modules in the system, NARY • each dependency module, NIND(I) with I=1, 2,...NARY, specified by NIND and NDEP • number of functional readiness configuration entries, NAV

TABLE 1. BREAKDOWN OF THE CARSRA INPUTS

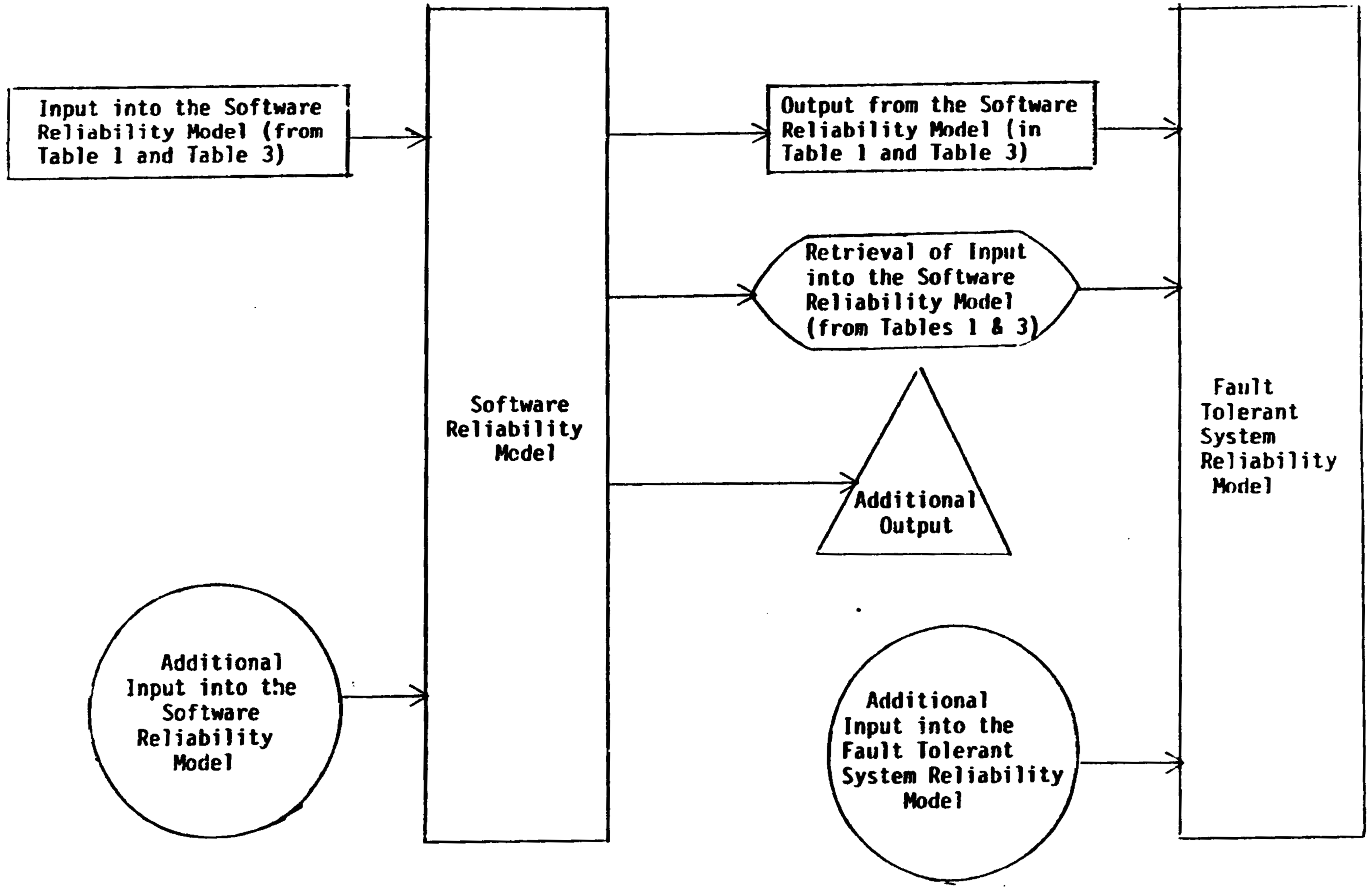

Input into the Software Reliability Model (from Table 1 and Table 3)
Additional Input into the Software Reliability Model
Software Reliability Model
Output from the Software Reliability Model (in Table 1 and Table 3)
Retrieval of Input into the Software Reliability Model (from Tables 1 & 3)
Additional Output
Additional Input into the Fault Tolerant System Reliability Model
Fault Tolerant System Reliability Model

FIGURE 1. INTERFACE DEFINITION FOR THE CARSRA AND CARE III MODELS

	Outputs from the Software Reliability Model				
	Dynamic Measurements			Static Measurements	
CARSRA Inputs	Hazard Rate, z(t)	Reliability Function, R(t)	Mean Time To Failure, MTTF	Total Number of Errors	Total Number of Remaining Errors
Transition rate matrix in failures per million hours, LMDA (NST, K, J) with J=1, 2,...NDIM and K=1, 2,.. ..(NDIM-1)	X				
Transitional readiness time span, AMT, and time increments, ADT			X		
Failure probability time span, FPMT, and time increment, FPDT		X	X		
Number of failed modules characterizing each configuration, NA (I, K)				X	
Number of stage failure patterns equivalent to system success, NOSCOF		X		X	

TABLE 2. RELATIONSHIP OF THE CARSRA INPUTS TO THE SOFTWARE RELIABILITY MODEL OUTPUTS

Outputs from the Software Reliability Model

	Dynamic Measurements			Static Measurements	
CARSRA Inputs	Hazard Rate, $z(t)$	Reliability Function, $R(t)$	Mean Time To Failure, MTTF	Total Number of Errors	Total Number of Remaining Errors
Success configurations, ICOF (I, J)		X		X	X
Accuracy Indicator, NACCUR		X			

TABLE 2. RELATIONSHIP OF THE CARSRA INPUTS TO THE SOFTWARE RELIABILITY MODEL OUTPUTS (Continued)

Input into the Software Reliability Model	Output from the Software Reliability Model	Input only into the Fault Tolerant System Reliability Model
• variable which defines if all of the fault handling models have exponential distributions only, MARKOV	• number of fault types to be included in the model, NFTYPS	• flag for outputting the moments single and double fault coverage functions, CVPRNT
• parameter for transition between the active state to the detected state, DEL(i)	• probability that a faulty operation will be successfully masked by the system, C(i)	• flag for a plot of the single and double fault coverage functions, CVPLOT
• parameter for transition from the active state to the active error, RHO(i)	• probability that a module detected as faulty in an active state is identified as a permanent fault and isolated from the system, PA(i)	• Y-axis scale for plotting coverage functions, IAXSCV
• parameter for transition from the error producing state to the detected state or to a single fault failure, EPS(i)	• probability that a module detected as faulty in a benign state is identified as a permanent fault and isolated from the system, PB(i)	• parameter governing the step doubling rule used in the solution, DBLDF
• indicator variable defining if the DEL parameter is for an IDELF(i)	• exponential rate for transition from an active state to a benign state, ALP(i)	• coverage function's truncation value, TRUNC
• indicator variable defining if the RHO parameter is for an exponential or uniform density, IRHOF(i)	• exponential rate for transition from a benign state to an active state, BET(i)	• option for the output printout, IRLPCD
• indicator variable defining if the EPS parameter is for an exponential or uniform density, IEPSF(i)	• number of fault types that a stage is subject to, NFCATS(x)	• option for the summary information to be plotted against time, RLPLOT
• number of stages in the system, NSTGES		• axes specification for the summary information plot, IAXSRL
		• number of equal steps that the flight time is divided into, NSTEPS
		• flag indicating whether or not the system fault tree is to follow, SYSFLG

TABLE 3. BREAKDOWN OF THE CARE III INPUTS

Input into the Software Reliability Model	Output from the Software Reliability Model	Input only into the Fault Tolerant System Reliability Model
• number of identical modules in stage number x, N(x) • minimum number of modules needed for stage ISTG to be operational, M(x) • operational configurations for stage x, NOP (i, x) • flight time for which the system is to be assessed, FT • time scale used for the flight time, ITBASE	• fault types specification for stage x, JTYP (j, x) • parameter ω of the Weibull fault occurrence rate for fault type j for stage x, OMG (j, x) • parameter λ of the Weibull fault occurrence rate for the fault type j for stage x, RLM (j, x)	• flag indicating whether or not a critical pairs fault tree is provided, CPLFLG • parameter used to limit the number of terms used in computing the coverage failure probability, PSTRNC • parameter used to limit the number of fault vectors used in computing the probability of system failure due to a lack of coverage, QPTRNC • parameter affecting the computation of the summary information, KWT • identification labels, TITLE

TABLE 3. BREAKDOWN OF THE CARE III INPUTS (Continued)

	Outputs from the Software Reliability Model				
	Dynamic Measurements			Static Measurements	
CARE III Inputs	Hazard Rate, z(t)	Reliability Function, R(t)	Mean Time To Failure, MTTF	Total Number of Errors	Total Number of Remaining Errors
Number of fault types to be included in the model, NFTYPS				X	X
Probability that a faulty operation will be successfully masked by the system, C(i)		X	X		
Probability that a module detected as faulty in an active state is identified as a permanent fault and isolated from the system, PA(i)	X	X			
Probability that a module detected as faulty in a benign state is identified as a permanent fault and isolated from the system, PB(i)	X	X			
Exponential rate for transition from an active state to a benign state, ALP(i)		X			
Exponential rate for transition from a benign state to an active state, BET(i)		X			
Number of fault types that a stage is subject to, NFCATS				X	

TABLE 4. RELATIONSHIP OF THE CARE III INPUTS TO THE SOFTWARE RELIABILITY MODEL OUTPUTS

CARE III Inputs	Outputs from the Software Reliability Model				
	Dynamic Measurements			Static Measurements	
	Hazard Rate, z(t)	Reliability Function, R(t)	Mean Time To Failure, MTTF	Total Number of Errors	Total Number of Remaining Errors
Fault types specification for stage x, JTYP (j, x)				X	X
Parameter ω of the Weibull fault occurrence rate for fault type j for stage x, OMG (j,x)	X				
Parameter λ of the Weibull fault occurrence rate for fault type j for stage x, RLM (j, x)	X				

TABLE 4. RELATIONSHIP OF THE CARE III INPUTS TO THE SOFTWARE RELIABILITY MODEL OUTPUTS (Continued)

Output from the Software Reliability Model	Input only into the Fault Tolerant System Reliability Model
• error probability of version #i, p_i • maximum number of versions allowed to be faulty at any crosscheck point, m^* • maximum number of versions allowed to be faulty in common mode, f^*	• number of active versions, n • correlation coefficient between #i and # j, $\rho_{i,j}$

TABLE 5. BREAKDOWN OF THE NVS INPUTS

	Outputs from the Software Reliability Model				
	Dynamic Measurements			Static Measurements	
NVS Inputs	Hazard Rate, $z(t)$	Reliability Function, $R(t)$	Mean Time To Failure, MTTF	Total Number of Errors	Total Number of Remaining Errors
Error probability of version #i, p_i	X				
Maximum number of versions allowed to be faulty at any crosscheck point, m^*		X		X	X
Maximum number of versions allowed to be faulty in common mode, f^*		X		X	X

TABLE 6. RELATIONSHIP OF THE NVS INPUTS TO THE SOFTWARE RELIABILITY MODEL OUTPUTS

4.0 CONCLUSION

With the interface definitions as described, the software reliability model is able to be used in a stand alone environment or in conjunction with the fault tolerant system reliability model. This setup is useful for error detection. The first stage in providing fault tolerance is to detect errors arising from the execution of the primary module. During its execution, the module will be subjected to the interface checks provided by the underlying system. These checks could detect the consequences of faults in the module and hence signal an exception.

5.0 BIBLIOGRAPHY

1. Anderson, T. and Lee, P. A., FAULT TOLERANCE Principles and Practice, Prentice-Hall International, Inc., Englewood Cliffs, New Jersey, 1981.

2. Bjurman, B. E. et al, "CARSRA: Computer Aided Redundant System Reliability Analysis Programmer's and User's Manual", Boeing Commercial Airplane Company, Report Number NASA-CR-145024, August 1976.

3. "Evaluation of Reliability Modeling Tools for Advanced Fault-Tolerant Systems", AIRLAB INTERFACE, A Progress Report, NASA Langley Research Center, Hampton, Virginia, December 1985.

4. Makam, Srinivas V., "Design Study of a Fault-Tolerant Computer System to Execute N-Version Software", University of California at Los Angeles, Technical Report No. CSD 821222, December 1982.

5. Ness, W. G., McCrary, W. C., Bridgman, M. S., Hitt, E. F., and Kenney, S. M., "Automated Reliability and Failure Effects Methods for Digital Flight Control and Avionic Systems, Volume I: Evaluation", Lockheed-Georgia Company and Battelle Columbus Laboratories, Columbus, Ohio, Report Number NASA-CR-166148, March 1981.

6. Prater, Shirley A., "Software Reliability Assessment Methods, Review of Studies of Software Reliability Models", Battelle Columbus Laboratories, Columbus, Ohio, October 1985.

7. Rose, D. M., Altschul, R. E., Manke, J. W., and Nelson, D. L., "CARE III User's Guide", Boeing Computer Services, Seattle, Washington (Prepared Under Contract Number NAS1-16900), January 1984.

8. Shooman, Martin L., SOFTWARE ENGINEERING Design/Reliability/Management, McGraw-Hill Book Company, New York, 1983.

TECHNICAL REPORT

on

FORMULATION OF THE SOFTWARE RELIABILITY MODEL

1.0 INTRODUCTION

A hierarchical software reliability model which predicts the reliability of software prior to its development is proposed. This model shall include both fault tolerant and fault intolerant software considerations. With this model, measurement of the reliability of software under development and identification of the data to be collected to make this evaluation shall be possible.

2.0 SOFTWARE CHARACTERISTICS

To handle both fault tolerant and fault intolerant software, the reliability model shall include single version software, N-version software, decision algorithm(s), recovery block(s), and acceptance test(s). The software characteristics of each of these design techniques are discussed in the following sections. In addition, when trying to specify software reliability, the principal concern in actually to describe the ways the software can be unreliable. Software reliability may be characterized by a profile that describes the modes of failure that the software can exhibit as a consequence of faults [DURHAM]. Therefore, the types of faults that are related to each of these design techniques are included in the following sections.

2.1 Single Version Software

This is a probabilistic model of deterministic or random events. Usually, the program execution is deterministic, while the development process is probabilistic. Some examples of faults that are characteristic in single version software and must therefore be accounted for are:

a. Incorrect specification
b. Misunderstood or unclear specification
c. Algorithmic error (sometimes called a computational or logic error)
d. Input data error
e. Program logic error
f. Output data error

2.2 N-Version Software

N-Version software is a fault-tolerant software technique which implements, usually in parallel, two or more versions that are functionally equivalent. These versions may be produced independently by separate programming teams or they may be made explicitly different through examination and subsequent forcing of differences into the versions [KELLY]. Nevertheless, when the alternate versions are compared, the faults should be distinguishable [HITT84]. Some of the faults associated with N-version software include:

a. Specification error
b. Performance error (due to incomplete, inconsistent, or ambiguous specifications)
c. Non-termination error
d. Algorithmic error
e. Input data error
f. Output data error

2.3 Decision Algorithm

The decision algorithm determines what the specific output should be. The decision algorithm may be a majority vote, a median select, a bit-by-bit comparison (with the number of bits that are to be compared or are significant specified), or an average [KELLY]. Some considerations to be made in the software design are:

a. The type of decision algorithm used;

b. The allowable range of discrepancy of each input from all other inputs to the decision algorithm; and

c. The data sensitivity of the decision algorithm.

2.4 Recovery Block

The recovery block method is a fault-tolerant software technique which provides alternate components which may be switched in (usually serially) to take the place of a faulty component that has been rejected by the acceptance test. These alternate components are designed independently from the main software component (the primary alternate) and generally only provide partial functionality of the software component, thus reducing it to a degraded, simpler mode. Prior to entering an alternate, the state of the process is restored to that current just before entry to the primary alternate [RANDELL]. Some examples of faults that occur in the software for recovery blocks include:

a. Specification error
b. Performance error
c. Non-termination error
d. Algorithmic error
e. Input data error
f. Output data error

2.4.1 Forward Recovery Block

A forward recovery block restores the system to a consistent state by compensating for inconsistencies found in the current state. For a single process, the forward recovery block technique requires a detailed knowledge of the extent of damage done and a strategy for fixing the inconsistencies [HITT86]. Therefore, for each data abstraction, exceptions shall be specified as a response to run-time attempts to violate its inherent invariant properties. These anticipated faults can be handled by forward recovery block techniques [HITT84 and CRISTIAN].

2.4.2 Backward Recovery Block

Backward recovery block techniques involve restoring the system to some previous known correct state (referred to as rollback) and restarting the computation from that point [HITT86]. Unanticipated faults, i.e., design faults, can be handled by a default exception handler using automatic backward recovery [HITT84 and CRISTIAN].

2.5 Acceptance Test

An acceptance test is a logical expression or algorithm which checks the acceptability of the results (or input) that are generated by a software component [RANDELL]. The faults that are associated with an acceptance test include:

a. Specification error
b. Performance error
c. Algorithmic error
d. Input data error
e. Output data error

2.6 Rollback

The rollback recovers the input state of the software to its condition prior to when an incorrect or faulty version was run. This resets the software to the input state necessary to run the next version. A rollback is used in connection with a recovery block and hybrid N-version software systems. Faults that are characteristic of rollback are:

a. Specification error
b. Input data error
c. Output data error
d. Unrecoverable state

2.7 Roll-Forward

The roll-forward is always used in connection with a forward recovery block. The roll-forward transfers the restored state obtained from the forward recovery block to a forward position in the system. The forward position for this transfer depends upon the state for which the forward recovery block has compensated. Faults associated with roll-forward include:

a. Specification error
b. Performance error
c. Input data error
d. Output data error

3.0 SOFTWARE INTERFACES

Software reliability is a probabilistic measure and is defined as the probability that a software error which causes discrepancies from specified requirements in a specified environment does not lead to a failure during a specified exposure period.

3.1 Inputs

The inputs to this software reliability model are the individual probabilistic reliability values (or safety, availability, or accuracy values, if desired) for the function blocks. These values are either obtained from the software reliability data base, estimated by the lines of code (and the language), derived experimentally by subjecting the function block's software to a number of test cases and counting the failures to determine a reliability value, or from lower (detailed) level models. The inputs should be real numbers with a range of $0.0 \leq$ probabilistic reliability value (or safety, availability, or accuracy value) ≤ 1.0.

3.2 Outputs

The output of the software reliability model is the overall probabilistic reliability value (or safety, availability, or accuracy values, if desired) of the closed loop block diagram. The output can also be for different levels within the hierarchical software reliability model, ranging from simple, high level block diagrams to complex, detailed block diagrams.

3.3 Communications

When first developed, a function block may be considered to be highly reliable, but if the software of that function is subsequently rarely used or tested, the confidence in that reliability value may be much lower than it would be with extensive use and testing. Two such examples are a backup bus controller and the auto land capability on some aircraft. (The auto land capability, on some aircraft, is checked only while the aircraft prepares for takeoff and then not again during the entire flight until it is actually needed for landing.)

4.0 SOFTWARE FUNCTIONS

The model is represented using control system notation for model representation. Each software module is considered to represent a transformation of input to output. While a signal flow graph could be used, a simulation diagram equivalent to the flow graph has been selected.

4.1 Menu Selection

Each module can be represented by a transfer function whose type is a unique icon. The software shall enable menu selection of the following icons:

a. Structure Icons

- single version software
- N-version software (the number of versions must be specified by the user)
- decision algorithm
- recovery block (the number of alternates must be specified by the user)
- acceptance test
- rollback
- roll-forward

b. Transfer Icons

- forward path
- positive feedback
- negative feedback
- positive feed-forward
- negative feed-forward

It shall be possible to place these icons along a display such that a block diagram is formed. Each of the structure icons shall represent a function in the software that is under development.

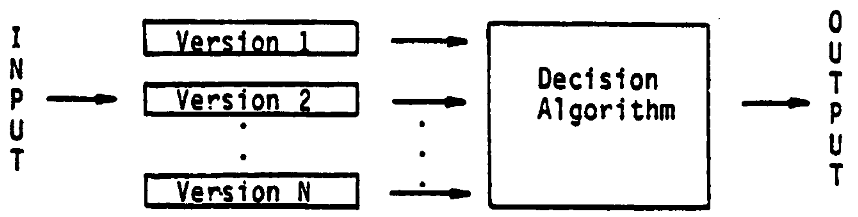


Figure 1. General Format for N-Version Software

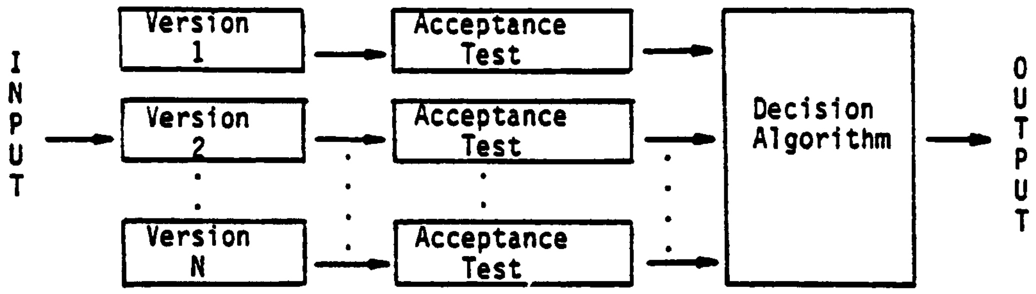


Figure 2. N-Version Software with Acceptance Tests

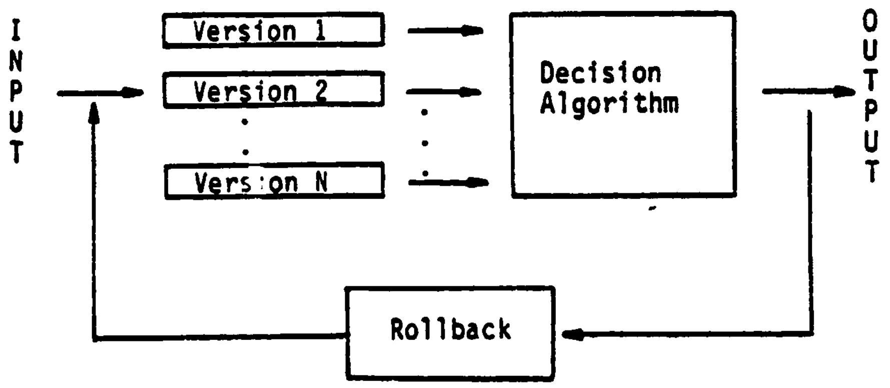


Figure 3. N-Version Software in Which Only x Versions are Used at a Time

4.1.1 Placement Requirements

The structure icons, given in Section 4.1.a, are listed as independent entities. However, when N-version software is chosen, it must be coupled with a decision algorithm in the general formats depicted in Figures 1-4.

Figure 3 represents an N-version software model in which only x of the versions are run at a time. If these x versions fail at the decision algorithm, then the software is "rolled back" (or restored to the original input state), and another x versions are run. This cycle will continue until the decision algorithm passes, the software "times out" (reaches its maximum time limit), or all of the versions have been run [SONERIU].

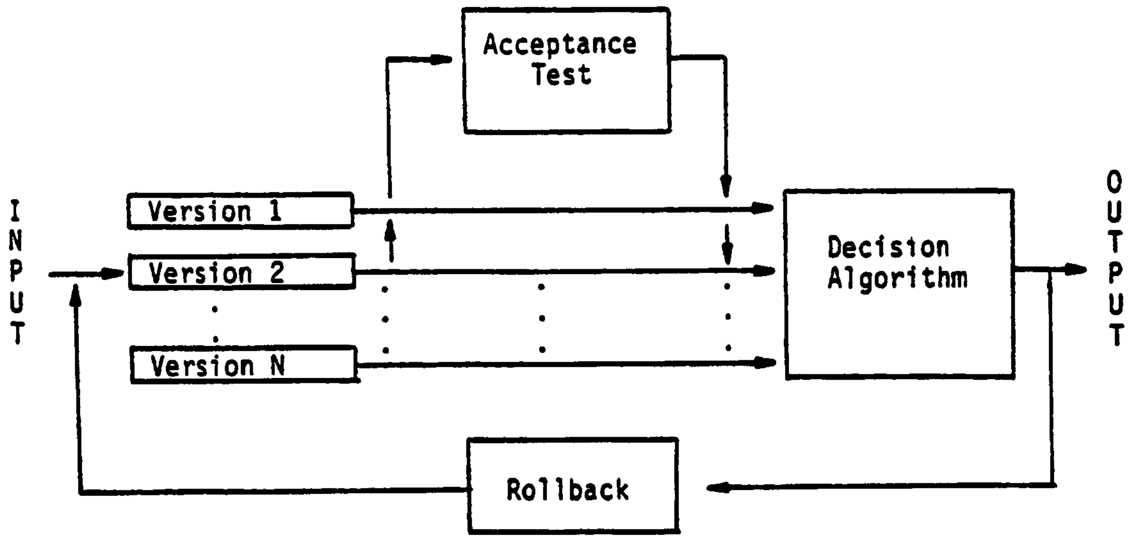


Figure 4. N-Version Software in Which the Outputs are Subjected to an Acceptance Test if the Decision Algorithm Fails [SCOTT]

When the recovery block is chosen, it should be used with an acceptance test in a format similar to:

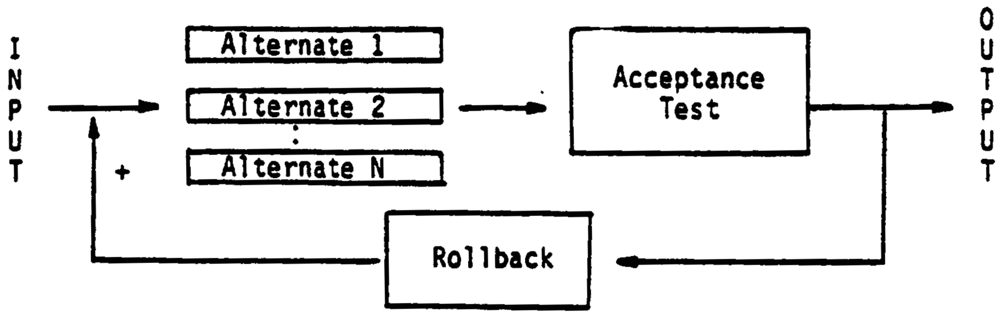


Figure 5. General Format of a Backward Recovery Block

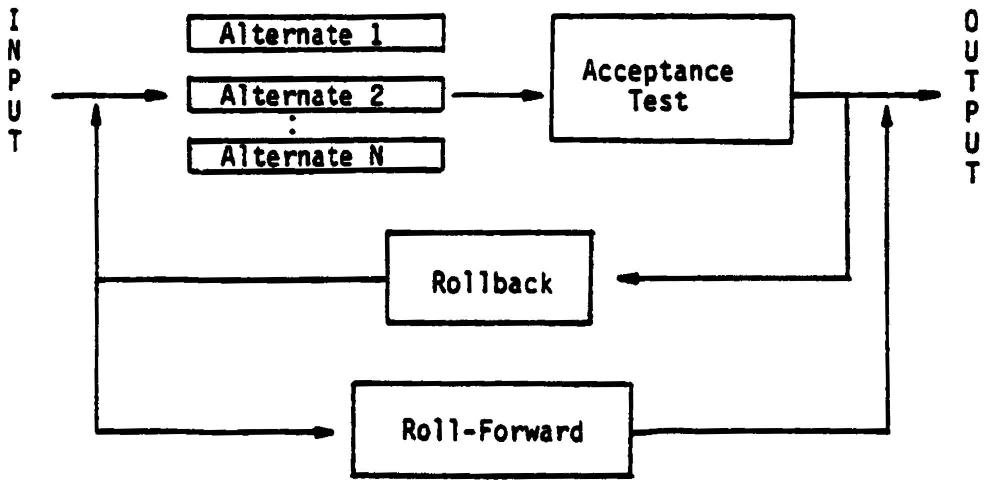


Figure 6. General Format of a Forward Recovery Block

A variation of the forward recovery block format might be:

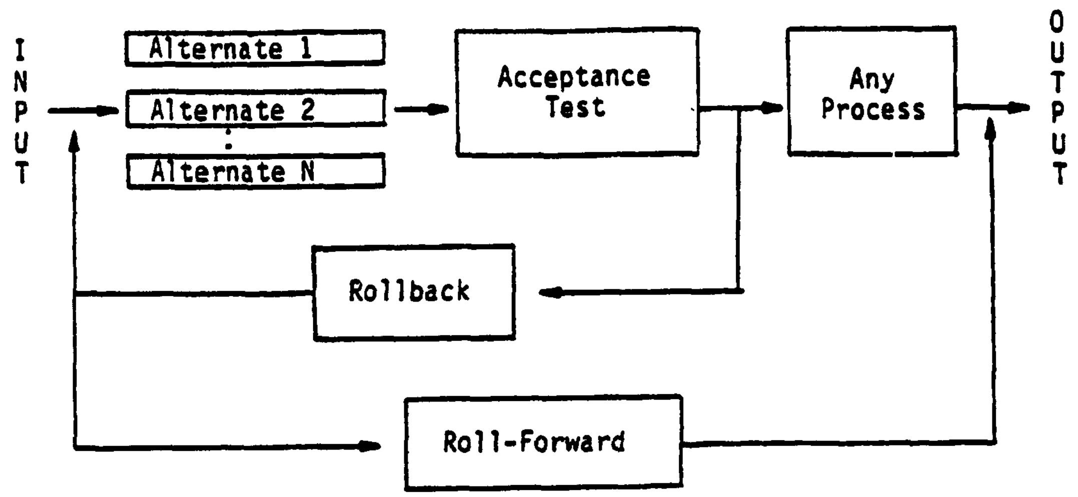


Figure 7. Alternative Format for a Forward Recovery Block

Section 4.1 lists the decision algorithm and acceptance test independently of the N-version software and recovery blocks to allow for the variations in the format. This permits the decision algorithm and acceptance test to be used independently, as well as in conjunction with their respective pairs [(N-version software and decision algorithm) and (recovery block and acceptance test)]. Keeping the decision algorithm and acceptance test as separate entities requires that each N-version software, decision algorithm, recovery block, and acceptance test module have individual reliability values. This should improve the accuracy of the transfer functions for this portion of the diagram since it will accommodate variations in the implementation of these concepts. (Reference Section 6.0).

4.2 Federal Aviation Administration (FAA) Function Criticality Categories

The system functions shall be classified as critical, essential, or non-essential, according to the effects of malfunctions or design errors. The categories are defined as:

a. Critical - Functions for which the occurrence of any failure condition or design error would prevent the continued safe flight and landing of the aircraft.

b. Essential - Functions for which the occurrence of any failure condition or design error would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions.

c. Non-Essential - Functions for which failures or design errors could not significantly degrade aircraft capability or crew ability.

The most critical function of a system will determine the category of the whole system unless that system has been partitioned into elements having different categories. Correspondingly, the software levels used throughout this report are Level 1, Level 2, and Level 3. The software level required for certification of functions is based upon the applicable criticality category. Level 1 is associated with the critical category, Level 2 with the essential category, and Level 3 with the non-essential category [RTCA].

4.3 Function Block Reliability

It shall be possible to determine the transfer function ("reliability") for the function blocks in the block diagram through the use of a software reliability model which addresses dynamic measurements. These transfer functions are often available through previous research and will consequently be furnished in the software reliability data base. (Reference Subtask 4.4.4, Define Data Required for the Software Reliability Data Base and Set Up the Data Base).

Furthermore, the FAA criticality categories will be supplied for each of the transfer functions to identify which of the function blocks or parameters strongly effect the overall system criticality. Any variation in the criticality of these function blocks would have a dramatic effect on the overall criticality estimate and its associated confidence level.

4.3.1 Detailed Function Blocks

This hierarchical software reliability model may be used with varying levels of detail [and consequently will provide varying degrees of accuracy (Reference Section 6.1.)]. Figure 8 gives a simple example of a possible situation. In this example, reliability values (probability of the software functioning correctly) may be substituted for each function block. This will permit the determination of the overall system reliability. (Reference Section 4.5.) However, this is a very high level model, and as such, the accuracy of the reliability values tend to be not as good as might be desired. Each of the software design techniques (single version software, N-version software, decision algorithm, recovery block, and acceptance test) can actually be broken down into more detailed function blocks, dependent upon the possible faults associated with the software. These faults were discussed in Sections 2.1 through 2.7.

A detailed diagram for the single version software function block and the decision algorithm function block are given in Figure 9. With reference to Sections 2.1 and 2.3, Tables 1 and 2, respectively, show the association between the identified faults and the portion of the diagram in Figure 2 in which they would occur. Therefore, the reliability of each of the detailed function blocks in Figure 8 is a probability of success for that portion [or 1.0 - (the probability that the associated fault(s) listed in Table 1 or 2 for the detailed function block will occur)].

4.3.2 Function Block States

The four possible states for any of the function blocks (detailed or not) are:

a. the function block fails (an error is detected in the function block) and it is corrupt (contains one or more error);
b. the function block passes, yet it is corrupt;
c. the function block fails and it is error free; and
d. the function block passes and it is error free.

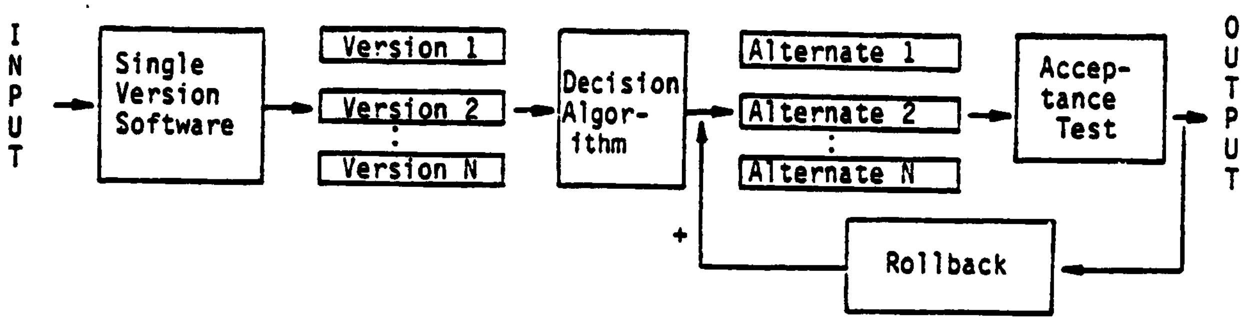


Figure 8. Simple Block Diagram Example

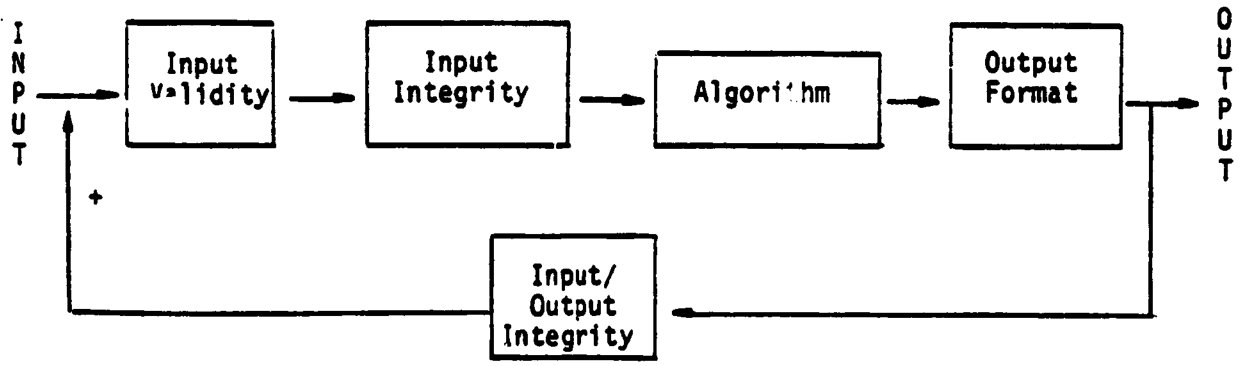


Figure 9. Detailed Diagram for the Single Version Software Function Block or the Decision Algorithm Function Block

Faults	Detailed Function Blocks				
	Input Validity	Input Integrity	Algorithm	Output Format	Input/ Output Integrity
Incorrect Specification	X	X	X	X	X
Misunderstood or Unclear Specification	X	X	X	X	X
Algorithmic Error	X	X	X	X	X
Input Data Error	X				X
Program Logic Error	X	X	X	X	X
Output Data Error				X	X

Table 1. Correlation of Faults with the Detailed Portions of the Single Version Software Function Block

	Detailed Function Blocks				
Faults	Input Validity	Input Integrity	Algorithm	Output Format	Input/ Output Integrity
Input Range Error	X	X			
Algorithmic Error	X	X	X	X	X

Table 2. Correlation of Faults with the Detailed Portions of the Decision Algorithm Function Block

A mathematical truth table for these states is given in Table 3.

State	Error Exists in the Function Block	Error Detected in the Function Block	Error Corrected in the Funtion Block	Safe	Reli-able	Avail-able
a	T	T	T	T	T	T
a	T	T	F	T	T*	F
b	T	F	N/A	F	F	F*
c	F	T	N/A	T	F	F
d	F	F	N/A	T	T	T

Key:

N/A = not applicable
* = True or False (the common interpretation is given)

Table 3. Truth Table of Function Block States

4.4 Software Reliability Data Base

A software reliability data base shall be established to store reliability values for the various function blocks identified in the software reliability model. These software reliability values will be collected from research performed and documented in technical reports. The use of these reliability values will provide a more accurate estimation of the software reliability and the confidence associated with this estimate.

4.5 System Reliability

The function blocks will each have an associated transfer function ("reliability"). The overall system reliability is determined via block diagram reduction techniques, thus giving the overall system transfer function.

In this software reliability model, the signal-flow diagram reduction technique is used to determine the overall system transfer function ("reliability"). The signal-flow diagram is useful in analyzing multiple-loop feedback systems and in determining the effect of a particular element or parameter in an overall feedback system, whereas the block diagram is useful in the design and analysis of sections of a feedback system. Block diagram reduction techniques become tedious and time consuming as the number of feedback paths increases. To solve complex problems, it is much simpler to use the theorems and properties of signal-flow graphs.

The equations used in this analysis follow S. J. Mason's theorems on the properties of signal-flow graphs. The general expression for the (closed loop) system transfer function using the signal-flow diagram reduction technique is given by

$$\text{Reliability} = \frac{\Sigma_K G_K \Delta_K}{\Delta}$$

where

$\Delta = 1 - \Sigma L_1 + \Sigma L_2 - \Sigma L_3 + \ldots + (-1)^n \Sigma L_n$,

L_1 = the gain of each closed loop in the graph,

L_2 = the product of the loop gains of any two non-touching closed loops,

L_3 = the product of the loop gains of any three non-touching closed loops,

L_n = the product of the loop gains of any n non-touching closed loops,

G_K = the gain of the Kth forward path,

Δ_K = the value of Δ for that part of the graph not touching the Kth forward path [SHINNERS].

The transfer function for N-version software and recovery blocks are dependent upon the number of versions or alternates (n). For N-version software, the transfer function is

$$1 - \prod_{i=1}^{Cn} (1 - \alpha_i)$$

with

n = the number of versions in the N-version software;

C(n,r) = the number of r combinations of an n element set;

Cn = C(n,z);

α = the product of reliabilities of the i-th combination required for success;

i = 1, 2, 3, ...Cn;

z = [(n/2) + 1] if n is an even number; and

z = [(n + 1)/2] if n is an odd number.

For a recovery block, the transfer function is

$$G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + \ldots.$$

with G_i = the reliability value for alternate i and
i = 1, 2, 3,...n.

The reliability values (transfer functions) for the hybrid N-version software and the recovery block will vary if not all of the n versions or n alternates are used. The above equations will give a higher reliability value than the actual situation in these cases. Sections 6.1.1 and 6.1.2 discuss the accuracy of the reliability values for the N-version software and recovery block and how they can be calculated to reflect the actual situation. (See Appendices I and II for some examples with these equations).

Although the transfer function for most of the function blocks is the reliability value, one exception to this is with rollback or any feedback block. For any feedback path, the transfer function of the equivalent block in the path is (1.0 - reliability value). (See Appendix III for a further explanation and proof). The second exception is with feed-forward paths. The transfer function of the equivalent block in a feed-forward path (for example, roll-forward) is (1.0 - reliability value). (Refer to Appendix IV for additional information.)

4.5.1 Simple, High Level Model Example

For a sample problem involving a simple, high level model, the example given in Figure 8 will be used. The Single Version Software will be a commonly used algorithm or process, and therefore, it will have a high reliability which is well documented and stored in the software reliability data base. For this example, the reliability will be 0.9991.

The N-Version Software will have three indepedent versions, running in parallel. This is a common form of N-Version Software, but not one with the highest reliability. Through the user's tests, it is determined that this function block will have a reliability of 0.994.

The Decision Algorithm will take an average of the outputs from the N-Version Software, excluding any version which does not meet the timing constraints. This type of Decision Algorithm has a high reliability since it does not have a range check or any other source for determining the validity of the output. This Decision Algorithm will not eliminate any erroneous outputs and will not detect the occurrence of correlated faults. The decision algorithm is simple (and consequently highly reliable), but it is not always the most desirable since its simplicity detracts from its capability of detecting errors. For this example, it is assumed that the reliability of the Decision Algorithm is 0.988.

The Recovery Block is a backward recovery block with a primary alternate and two additional, extremely simplified alternates. The Recovery Block is of a common form and its reliability can be obtained from the software reliability data base. For this example, it will have a reliability value of 0.97.

The Acceptance Test is an output format check. This is a simplistic algorithm which is commonly used in various models. The reliability, as determined from the software reliability data base, will be 0.9997.

Finally, the Rollback recovers the input state of the software to its condition upon entry to the Recovery Block. This is a retrieval of the data from its memory location. The reliability of this common form of Rollback will be assumed to be available in the software reliability data base. For this example, the reliability is 0.9999. This makes the transfer function for the Rollback equal to (1.0 - 0.9999). (Reference the final paragraph in Section 4.5.)

Hence, for this example,

$$L_1 = (0.97) \times (0.9997) \times (+1.0 - 0.9999) = 0.0000969709$$

$$L_2 \text{ through } L_n = 0$$

$$G_1 = (0.9991) \times (0.994) \times (0.988) \times (0.97) \times (0.9997) = 0.951467$$

$$G_2 \text{ through } G_K = 0$$

$$\Delta_1 = 1$$

$$\Delta = 1 - (+0.0000969709) = 0.99990302$$

Therefore,

$$\text{Reliability} = \frac{(0.951467) \times (1)}{(0.99990302)} = 0.951559$$

$$\text{Reliability} = 0.95.$$

4.5.2 Complex, High Level Model Example

The complex, high level model example will be as shown in Figure 10. Block (1) is a Single Version Software block. For this example, it will be a simple algorithm with a reliability value of 0.998.

Block (2) is N-Version Software with nine independent versions in which only three versions are run at a time. (This type of N-Version Software was shown in Figure 3). For this example, it is assumed that the reliability value for the N-Version Software is 0.999.

Block (3) is the Decision Algorithm for the N-Version Software. In this example, the Decision Algorithm will be a median select with a reliability value of 0.983.

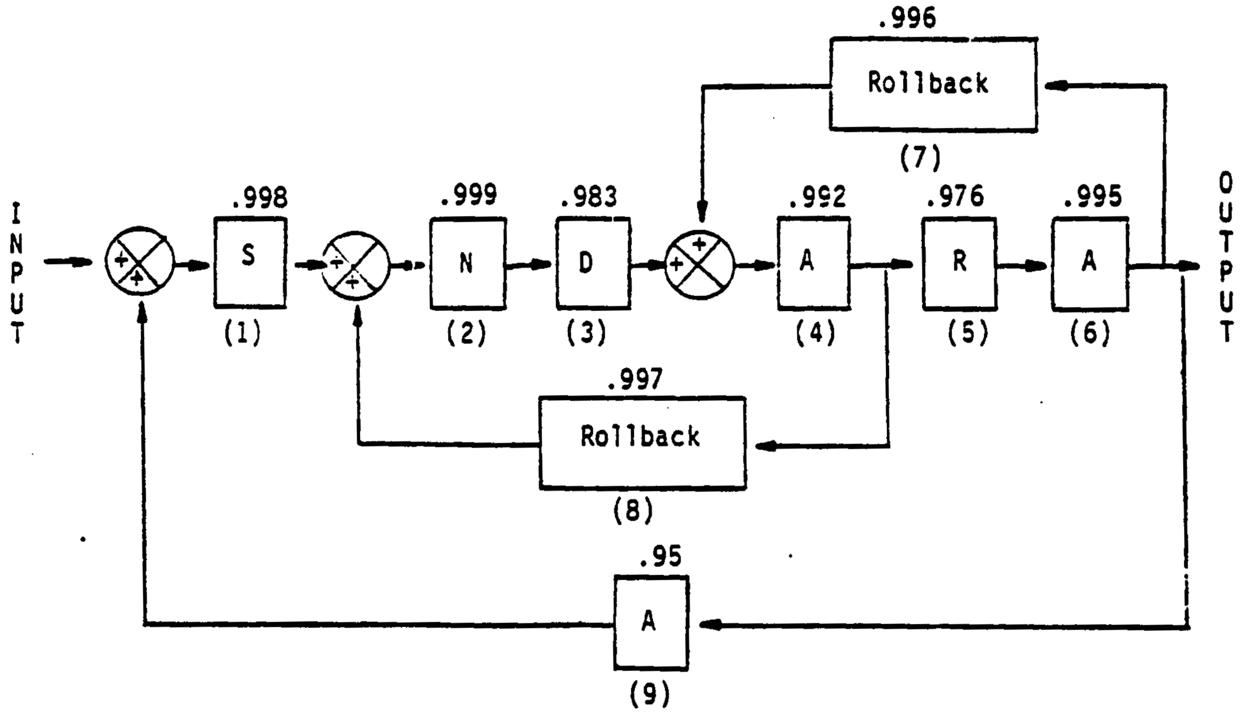
Block (4) is an Acceptance Test which will check that the range of the output from the Decision Algorithm is correct. If the Decision Algorithm failed, then the software will Rollback after the Acceptance Test. Similarly, if the Acceptance Test fails, then the software will Rollback to the N-Version Software. The input to the Acceptance Test will be stored to accommodate for the Rollback from the Recovery Block. For this example, the reliability value for the Acceptance Test will be 0.992.

Block (5) is a Recovery Block of the common form with a primary alternate and two additional alternates. For this example, the reliability value of the Recovery Block will be 0.976.

Block (6) is the Acceptance Test for the Recovery Block. In this example, it is assumed that the reliability value for this Acceptance Test is 0.995.

The Rollback for the Backward Recovery Block is given in block (7). This is a retrieval of the data that was stored prior to entry into Acceptance Test #1. For this example, the reliability value for this Rollback is 0.996. Thus, the transfer function for this block is (1.0 - 0.996).

The Rollback for the N-Version Software is given in block (8). For this example, the reliability value is 0.997. Consequently, the transfer function is (1.0 - 0.997).



Key:

(1) Single Version Software
(2) N-Version Software
(3) Decision Algorithm
(4) Acceptance Test #1
(5) Recovery Block
(6) Acceptance Test #2
(7) Rollback #1 -- Backward Recovery Block
(8) Rollback #2 -- N-Version Software in Which Only x Versions are Used at a Time
(9) Acceptance Test #3

Figure 10. Complex, High Level Model Example

Finally, block (9) is an Acceptance Test which checks the final output against the input. For this example, Acceptance Test #3 will have a reliability value of 0.95, giving a transfer function of (1.0 - 0.95).

Hence, for this example,

$$\text{Closed Loop 1} = (0.999) \times (0.983) \times (0.992) \times (1.0 - 0.997)$$

$$= 0.0029225$$

$$\text{Closed Loop 2} = (0.992) \times (0.976) \times (0.995) \times (1.0 - 0.996)$$

$$= 0.0038534$$

$$\text{Closed Loop 3} = (0.998) \times (0.999) \times (0.983) \times (0.992) \times (0.976) \times (0.995) \times (1.0 - 0.95)$$

$$= 0.0472068$$

$$\Sigma L_1 = (\text{Closed Loop 1}) + (\text{Closed Loop 2}) + (\text{Closed Loop 3})$$

$$= 0.0029225 + 0.0038534 + 0.0472068$$

$$= 0.0539827$$

L_2 through $L_n = 0$

$$G_1 = (0.998) \times (0.999) \times (0.983) \times (0.992) \times (0.976) \times (0.995)$$

$$= 0.944135$$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$$\Delta = 1 - (0.0539827) = 0.9460173$$

Therefore,

$$\text{Reliability} = \frac{(0.944135) \times (1)}{(0.9460173)} = 0.9980103$$

Reliability = 0.998.

4.5.3 Simple, Detailed Level Model Example

An example of a simple, detailed level model is given in Figure 9. For this example, the structure icons would all be single version software and the transfer icons would be either forward path or positive feedback (Reference Section 4.1.). Hence, an equivalent block diagram for this detailed block diagram is given in Figure 11.

In Figure 11, block (1) is a Single Version Software block which checks the input set. This is a common process, so the reliability value will be assumed to be available in the software reliability data base. For this example, the reliability value for block (1) is assumed to be 0.98.

Block (2), a Single Version Software block, will represent an input integrity check. It will be assumed that this algorithm is common and can consequently be found in the software reliability data base. For this example, the reliability value will be 0.97.

Block (3) is a Single Version Software block which performs an algorithm. For this example, the algorithm will be a simple one. Therefore, the reliability value for this transfer block will be assumed to be 0.992.

Block (4) will represent an output format check, performed by a Single Version Software block. For this example, the reliability value for this Single Version Software will be 0.996.

The final Single Version Software block, block (5), will be used to perform an input/output integrity check in which the output is checked against the input to verify the integrity of the input data. For this example, the reliability value for block (5) will be 0.964. Hence, the transfer function for this block will be (1.0 - 0.964).

Thus, for this example,

$$\begin{aligned} L_1 &= (0.98) \times (0.97) \times (0.992) \times (0.996) \times (1.0 - 0.964) \\ &= 0.033812 \end{aligned}$$

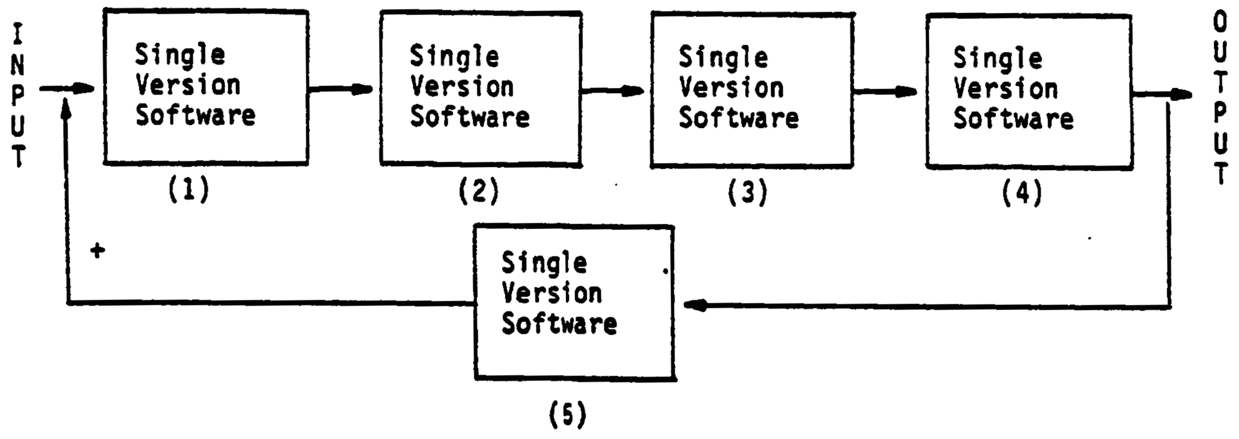


Figure 11. Equivalent Diagram Indicating the Structure Icons to be Used in the Detailed Diagram for a Single Version Software Function Block

L_2 through $L_n = 0$

$G_1 = (0.98) \times (0.97) \times (0.992) \times (0.996)$

$= 0.9392232$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1 - (0.033812) = 0.966188$

Therefore,

$$\text{Reliability} = \frac{(0.9392232) \times (1)}{(0.966188)} = 0.9720916$$

Reliability = 0.97.

4.6 Safety

Safety is concerned with the state in which the function block fails (an error is detected), but the function block is error free. Although a function block is considered to be unsafe when the system is unreliable, safety also covers this extra state. Hence,

safety = [(the probability that an error exists and it is detected) + (the probability that no error exists)]

or

safety = [1 - (the probability that an error exists and it is not detected)]

while reliability = [(the probability that an error exists and it is detected) + (the probability that no error exists and no error is detected)]

or

reliability = {1 - [(the probability that an error exists, but the error is not detected) + (the probability that no error exists, but an error is detected)]}

Therefore, safety $\geq$ reliability.

Actually, the software reliability model can be used to determine the safety of the high level models. Instead of usin. reliability values for each function block, in the determination of the overall transfer function, if the safety value for each of the function blocks is used in the calculations, the resultant value will be the safety of the overall system.

4.7 Availability

As with reliability and safety, availability can be determined through the use of the software reliability model. To do so, the availability values should be used for each function block in the determination of the overall transfer function. By using availability values instead of reliability values, the resultant value will be the availability of the overall system.

The availability values are determined as follows:

availability = [(the probability that an error exists, it is detected, and it is corrected) + (the probability that no error exists and no error is detected)]

or

availability = {1 - [(the probability that an error exists, it is detected, but it is not corrected) + (the probability that an error exists and no error is detected) + (the probability that no error exists and an error is detected)]}

Therefore, $0 \leq$ availability ≤ 1.0. By comparison to reliability and safety, availability $\leq$ reliability $\leq$ safety.

5.0 TIMING CONSTRAINTS

If a software component should execute in 1 msec., a time could detect software faults that cause the execution time to exceed 1 msec. Many of the reliability model's timing constraints deal with the fault tolerant portions where the entire process (fault detection, damage assessment, recovery, and fault treatment) must take place fast enough to satisfy real-time requirements. "No matter which fault tolerant software method is used, real-time systems must arrive at a consistently correct solution within the time frame determined by the control system dynamics. Failure can occur due to excessively long response times, e.g., the system goes unstable since the hard deadlines for code execution are missed."[HITT86]

6.0 ACCURACY CONSTRAINTS

The accuracy and reliability of the N-version software, decision algorithm, recovery block, and acceptance test are dependent upon the way in which these concepts are implemented. For example, N-version software may be implemented as:

a. two independent versions
b. three indpendent versions
c. more than three independent versions
d. an N-version software model in which only x versions are run at a time, and if these versions fail for some reason, then x or less of the remaining (N-x) versions are run. (This is depicted in Figure 3.)
e. an N-Version model in which a combination of x versions are run, and if these x versions fail for some reason, a different combination of x versions is run, and so on. (NOTE: This concept is different from item d, above, because this implementation groups different combinations of x versions. Item d, however, uses x versions, and if they fail, the x versions are in essence thrown out and a completely new group of x versions are used; not a new combination of x versions, but x completely new versions.)

Some of the differences which affect the reliability and accuracy of the decision algorithm are:

a. majority vote;
b. median select;
c. average; and
d. the decision algorithm only considers those values which are in certain range and then it uses one of the methods (a, b, or c), above.

The recovery block's accuracy and reliability depend on the following items (to name a few):

a. backward
   i.) how far it rolls back; and
   ii.) the number of alternates available.
b. forward
   i.) how far it rolls forward; and
   ii.) the accuracy of the value(s) assigned prior to the roll.

Some of the concepts that affect the accuracy and reliability of the acceptance test depend on:

a. the range of the values accepted;
b. the rate of change determination for the variables; and
c. the format of the data.

Needless to say, all of these implementation characteristics must be considered and will affect the accuracy of the reliability values. Sections 4.1 and 4.1.1 discuss how the software reliability model is set up to accommodate the implementation variations. With this software reliability model design, the accuracy is improved since these considerations can be taken into account in the assignment of reliability values (or accuracy -- see Sections 6.1 and 6.1.1 through 6.1.3 -- or safety or availability values) to the function blocks.

6.1 Accuracy

The accuracy of the software reliability model will depend upon the accuracy of the individual values that are used as the transfer functions for each of the function blocks. In most cases, the accuracy of a model with detailed function blocks will be better than the high level software reliability model since the accuracy of the individual transfer functions will be improved (Reference Section 4.3.1.).

The accuracy of the reliability values that are used for the function blocks' transfer functions will depend upon the method used to obtain such values. If the values are obtained from the software reliability data base, then the accuracy of these values are indicated in the technical reports for the research method that determined the values. If the values are obtained through the use of a different software reliability model, then the accuracy is dependent on the type of model used. Regardless, accuracy values can be obtained for any and all of the transfer functions, and therefore, an accuracy for the overall software reliability value can be calculated.

6.1.1 Accuracy of the Hybrid N-Version Software

The accuracy of the hybrid N-version software reliability value (or safety or availability values) depends upon the number of versions that are actually used. (Reference Figure 3 for an example of a hybrid N-version software.) Usually, if the software is run and only y of the n versions are used, then the reliability value (or safety or availability values) has been overrated by considering the additional (n-y) versions in the calculation of the transfer function for the hybrid N-version software. Certainly, if it is known that only y of the n versions are actually being used, then the calculation of the transfer function for the N-version software should only include those y versions. (See Appendix I, Example 3, for a demonstration of this accuracy effect.)

6.1.2 Accuracy of the Recovery Block

The recovery block, by definition, consists of n alternates. Of these n alternates, only one is run at a time, and only if that alternate fails will the software rollback and run the next alternate. Hence, if less than the n alternates are actually used, the reliability (or safety or availability) of the recovery block will generally decrease. Furthermore, this decrease in the recovery block's transfer function (reliability, safety, or availability value) will cause a decrease in the overall software reliability value (or safety or availability value), calculated with the software reliability model. (See Appendix II for some recovery block calculations that address the effect on accuracy when fewer than the n alternates are actually used.)

6.1.3 Accuracy Example for the Software Reliability Model

To demonstrate the use of the software reliability model for an accuracy calculation, the simple block diagram shown in Figure 8 and the example in Section 4.5.1 will be used. The first step in this determination is to assign accuracy values to each of the blocks. For this example, the following values will be used:

Function Block	Accuracy Value	Transfer Function
Single Version Software	± 0.0002	0.9998
N-Version Software	± 0.001	0.999
Decision Algorithm	± 0.003	0.997
Recovery Block	± 0.005	0.995
Acceptance Test	± 0.0001	0.9999
Rollback	± 0.00005	0.00005

Table 4. Accuracy Values for the Simple Block Diagram Example

These accuracy values reflect the accuracy of the reliability values that are used in the simple, high level model example.

Opposite of the software reliability model calculations, the transfer functions for the blocks are (1.0 - $|$accuracy value$|$), except for rollback, roll-forward, the equivalent block within a feedback loop, or the equivalent block within a feed-forward path, which use the absolute value of the accuracy value. The respective transfer functions are listed in Table 4.

Hence, for this example,

$L_1 = (0.995) \times (0.9999) \times (0.00005) = 0.0000497$

L_2 through $L_n = 0$

$G_1 = (0.9998) \times (0.999) \times (0.997) \times (0.995) \times (0.9999)$
$= (0.9907257)$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1 - 0.0000497 = 0.9999503$

Therefore,

$$\text{Accuracy} = 1 - \left| \frac{(0.9907257) \times (1)}{(0.9999503)} \right|$$

$$= 1 - \left|0.9907749\right| = \pm\ 0.0092251$$

$$\text{Accuracy} = \pm\ 0.00923.$$

7.0 RESPONSE TO UNDESIRED EVENTS

The software reliability model described herein assumes independence between the function blocks. This model neglects the existence of:

a. multiple faults which produce dissimilar outputs but are manifested by the same input conditions, or

b. related software design faults causing identical incorrect outputs.

The errors that are manifested by these faults are known as coincident errors and cause a degradation in the reliability (or safety or availability). Therefore, to improve the accuracy of the software reliability model, the coincident errors must be considered. This might be done with an analysis similar to that suggested by Dave E. Eckhardt, Jr. and Larry D. Lee. The analysis makes the assumptions that (1) the input series $X_1, X_2, \ldots,$ is stationary and independent and (2) the versions of software components are designed independently [ECKHARDT].

When evaluating the probability of coincident errors, the area of concern is the N-version software. This analysis is interested in the probability that z or more of the functions fail at the same time, with $z = (n/2)$ if n is even and $z = [(n + 1)/2]$ if n is odd. The following analysis will give a conservative estimate (maximum possible) of the probability of coincident errors for the N-version software. This value might be subtracted from the transfer function of the N-version software block to produce a conservative value (minimum) of the reliability (or safety or availability) of the N-version software and consequently a conservative estimate (minimum) of the overall software reliability value (or safety or availability value).

The following equation gives the maximum probability of coincident errors (E).

$$E = \binom{n}{z}^{*}(1 - G_L)^{z} + \binom{n}{z+1}^{*}(1 - G_L)^{z+1} + \binom{n}{z+2}^{*}(1 - G_L)^{z+2} + \ldots + \binom{n}{n}^{*}(1 - G_L)^{n}$$

with

n = the number of versions in the N-version software;

G_i = the reliability (or safety or availability) value for version i;

$i = 1, 2, \ldots n$;

G_L = the largest reliability (or safety or availability) value among the group of r versions being evaluated;

$z = (n/2)$ if n is an even number;

$z = [(n + 1)/2]$ if n is an odd number;

$\binom{n}{r}$ = the number of r combinations of an n element set; and

$\binom{n}{r}^{*}$ = the actual r combinations of $(1 - G_i)$ values for the different versions in an n element set of versions.

(See Appendix I for some examples which consider the effect of coincident errors.)

8.0 ASSUMPTIONS

The assumptions that are frequently made with the various software methods are described below, along with the reasons for such assumptions. These assumptions are logically grouped below the corresponding software to benefit the reader. In the development of this software reliability model, it is assumed that the software of the function blocks will have complete probabilistic independence. However, Section 7.0 tries to accommodate for any ill effects that result from this basic assumption.

<u>Single Version Software</u>

a. Errors are not always corrected when detected and errors may be spawned when correcting errors.

b. The time to remove a failure is considered to be negligible and is ignored.

c. Inputs which exercise the program are randomly selected.

d. The failure rate at any time is proportional to the current number of errors remaining in the program [PRATER].

<u>N-Version Software</u>

a. To benefit from increased reliability, N-version assumes the probability of a common fault among the versions is extremely low.

b. When a fault is determined, the damage incurred is limited to the encapsulation of the individual software versions and the overall function that the versions are performing.

<u>Decision Algorithm</u>

a. For a majority vote, it is assumed that damage will be limited to the versions in the minority when the decision algorithm is invoked.

b. It is possible for a majority vote to yield an incorrect result if a majority of the inputs are incorrect.

<u>Recovery Block</u>

a. Faults will manifest themselves within a recovery region.

b. The alternate versions of software components are independent such that correlated faults are either eliminated or reduced to an acceptably low level.

c. The n alternate blocks are independent from the acceptance tests [HITT84].

<u>Acceptance Test</u>

a. The acceptance test will recognize the faults.

9.0 REFERENCES

[AVLA86]	Avizienis, A. and Laprie, J. C., "Dependable Computing: From Concepts to Design Diversity", to be published in the IEEE Proceedings.
[CRISTIAN]	Cristian, F., "Exception Handling and Software Fault Tolerance", IEEE Transactions on Computers, Volume C-31, Number 6, June 1982.
[DURHAM]	Durham, Ivor and Shaw, Mary, "Specifying Reliability as a Software Attribute", Carnegie-Mellon University, Report Number CMU-CS-82-148, December 6, 1982.
[ECKHARDT]	Eckhardt, Dave E. Jr. and Lee, Larry D., "A Theoretical Basis for the Analysis of Redundant Software Subject to Coincident Errors", NASA Langley Research Center, NASA Technical Memorandum 86369, January 1985.
[HITT86]	Hitt, Ellis F., "Software Fault-Tolerance (Task D-1), Battelle Columbus Division, Columbus, Ohio, January 9, 1986, pp. 1, 5-7, and 20-25.
[HITT84]	Hitt, Ellis F., Webb, Jeffrey J., Bridgman, Michael S., "Comparative Analysis of Fault-Tolerant Software Design Techniques", Prepared Under Contract Number NAS1-17412, February 15, 1984, pp. 11-17, 30-35, 37, 40-44, and 68-90.
[KELLY]	Kelly, John P.J., "Specification of Fault-Tolerant Multi-version Software: Experimental Studies of a Design Diversity Approach", UCLA Technical Report, 1982.
[MCGARRY]	McGarry, Frank, Page, Jerry, Eslinger, Suellen, Church, Victor, and Merwarth, Phillip, <u>Recommended Approach to Software Development</u>, National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, Maryland, Report Number SEL-81-205, April 1983.
[PRATER]	Prater, Shirley A., "Software Reliability Assessment Methods, Review of Studies of Software Reliability Models", Battelle Columbus Division, Columbus, Ohio, October 1985.
[RANDELL]	Randell, B., "System Structure for Software Fault Tolerance", IEEE Transactions on Software Engineering, Volume SE-1, Number 2, June 1975.
[RTCA]	<u>Software Considerations in Airborne Systems and Equipment Certification</u>, Radio Technical Commission for Aeronautics, Report Number RTCA/DO-178A, March 22, 1985, pp. 13, 14, 20, and 42.

[SCOTT] Scott, Roderick Keith, "Data Domain Modeling of Fault-Tolerant Software Reliability", North Carolina State University, Raleigh, North Carolina, 1983, pp. 22-27, 37-39, and 42-45.

[SHINNERS] Shinners, Stanley M., Modern Control System Theory and Application, Addison-Wesley Publishing Company, Inc., U.S.A., 1978, pp. 46-56.

[SHOOMAN] Shooman, Martin L., SOFTWARE ENGINEERING Design/Reliability/ Management, McGraw-Hill Book Company, New York, 1983, pp. 297-300 and 416-425.

[SONERIU] Soneriu, M.D., "A Methodology for the Design and Analysis of Fault-Tolerant Operating Systems", PhD Dissertation, Illinois Institute of Technology, Chicago, Illinois, 1981.

[TOUL76] "Definition of the Pilot-Project on Computer System Dependability", Joint Report UPS-LSI/ONE RA-CERT/CNRS-LAAS, Toulouse, France, January 1976 (in French).

APPENDIX I
N-VERSION SOFTWARE CALCULATIONS

The transfer function for N-version software is given by the following equation:

$$1 - \prod_{i=1}^{Cn} (1-\alpha_i)$$

with n = the number of versions in the N-version software;

Cn = C(n,z) = the number of z combinations of the n element set;

α_i = the product of reliabilities of the i-th combination required for success; and

i = 1, 2,....Cn.

z = [(n/2) + 1] if n is an even number

z = [(n+1)/2] if n is an odd number

The following examples utilize this equation.

Example 1

Using the block diagram shown in Figure 1, a system with N-version software whose outputs go into a decision algorithm will be analyzed. For this example, the N-version software will consist of five versions. The reliability values for the five versions and the decision algorithm are given in Table 5.

Software Component	Reliability Value
Version 1	0.77
Version 2	0.82
Version 3	0.65
Version 4	0.91
Version 5	0.89
Decision Algorithm	0.997

Table 5. Reliability Values for the Software Components in the Figure that Represents the General Format for N-Version Software

To determine the overall software reliability value for this block diagram, the transfer function for the N-version software, which is dependent upon the number of versions (in this example $n = 5$), must first be determined. Hence, the transfer function for the N-version software (NVS) is

$$NVS = [1 - (1-G_1G_2G_3)(1-G_1G_2G_4)(1-G_1G_2G_5)(1-G_1G_3G_4)(1-G_1G_3G_5) \times (1-G_1G_4G_5)(1-G_2G_3G_4)(1-G_2G_3G_5)(1-G_2G_4G_5)(1-G_3G_4G_5)]$$

By substituting in the appropriate reliability values,

$$NVS = 1 - [1-(0.77)(0.82)(0.65)][1-(0.77)(0.82)(0.91)] \times [1-(0.77)(0.82)(0.89)][1-(0.77)(0.65)(0.91)] \times [1-(0.77)(0.65)(0.89)][1-(0.77)(0.91)(0.89)] \times [1-(0.82)(0.65)(0.91)][1-(0.82)(0.65)(0.89)] \times [1-(0.82)(0.91)(0.89)][1-(0.65)(0.91)(0.89)]$$

$$= [1 - (1-0.41041)(1-0.574574)(1-0.561946)(1-0.455455) \times (1-0.445445)(1-0.623623)(1-0.48503)(1-0.47437) \times (1-0.664118)(1-0.526435)]$$

$$= [1 - (0.58959)(0.425426)(0.438054)(0.544545)(0.554555) \times (0.376377)(0.51497)(0.52563)(0.335882)(0.473565)]$$

$$= 1 - 0.0005377 = 0.9994623$$

$$NVS = 0.99946$$

By applying the software reliability model in Section 4.5,

L_1 through $L_n = 0$
$G_1 = (0.99946) \times (0.997) = 0.9964616$
G_2 through $G_K = 0$
$\Delta_1 = 1$
$\Delta = 1$

Therefore,

$$\text{Reliability} = \frac{(0.9964616) \times (1)}{(1)} = 0.9964616$$

$$\text{Reliability} = 0.996.$$

The probability of coincident errors (E) should be determined and subtracted from the transfer function for the N-version software to improve the accuracy of the reliability value of the N-version software and the accuracy of the overall software reliability value. (Reference Section 7.0.) For this example,

$$E = \binom{5}{3}^{*} (1-G_L)^3 + \binom{5}{4}^{*} (1-G_L)^4 + \binom{5}{5}^{*} (1-G_L)^5.$$

The groups for $\binom{5}{3}^{*}$ would be

$G_1G_2G_3$, $G_1G_2G_4$, $G_1G_2G_5$, $G_1G_3G_4$, $G_1G_3G_5$, $G_1G_4G_5$, $G_2G_3G_4$, $G_2G_3G_5$, $G_2G_4G_5$, and $G_3G_4G_5$,

with respective G_L values being

G_2, G_4, G_5, G_4, G_5, G_4, G_4, G_5, G_4, and G_4.

The G_L values can be grouped as $G_2 + 6 \times G_4 + 3 \times G_5$.

The groups for $\binom{5}{4}^{*}$ would be

$G_1G_2G_3G_4$, $G_1G_2G_3G_5$, $G_1G_2G_4G_5$, $G_1G_3G_4G_5$, and $G_2G_3G_4G_5$,

with the respective G_L values being G_4, G_5, G_4, G_4, and G_4. This gives

$4 \times G_4 + G_5$.

The group for $\binom{5}{5}^{*}$ is $G_1G_2G_3G_4G_5$ with $G_L = G_4$.

Therefore,

$$E = (1-G_2)^3 + 6 \times (1-G_4)^3 + 3 \times (1-G_5)^3 + 4 \times (1-G_4)^4 + (1-G_5)^4 + (1-G_4)^5$$
$$= (1-0.82)^3 + 6 \times (1-0.91)^3 + 3 \times (1-0.89)^3 + 4 \times (1-0.91)^4 + (1-0.89)^4 + (1-0.91)^5$$
$$= (0.18)^3 + 6 \times (0.09)^3 + 3 \times (0.11)^3 + 4 \times (0.09)^4 + (0.11)^4 + (0.09)^5$$
$$= 0.005832 + 0.004374 + 0.003993 + 0.0002624 + 0.0001464 + 0.0000059$$
$$E = 0.0146137.$$

Subtracting E from the transfer function for the N-version software gives

$$NVS = 0.9994623 - 0.0146137 = 0.9848486$$
$$NVS = 0.98485.$$

L_1 through $L_n = 0$

$G_1 = (0.98485) \times (0.997) = 0.9818955$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1$

Hence, the adjusted Reliability value is

$$\text{Reliability} = \frac{(0.9818955) \times (1)}{(1)} = 0.9818955$$

$$\text{Reliability} = 0.982.$$

<u>Example 2</u>

This example will evaluate the overall reliability for the system shown in Figure 2. In this example, the N-version software will consist of four versions. Each of the outputs from the N-version software are submitted to an acceptance test (the identical acceptance test is used for all four versions), and then the outputs from the acceptance test are input to the decision algorithm. The reliability values for each of the software components are given in Table 6.

Software Component	Reliability Value
Version 1	0.86
Version 2	0.79
Version 3	0.94
Version 4	0.68
Acceptance Test	0.98
Decision Algorithm	0.93

Table 6. Reliability Values for the Software Components in the Figure that Represents the N-Version Software with Acceptance Tests

First, the transfer function for the N-version software (NVS) must be determined. In this example,

$$NVS = [1 - (1-G_1G_2G_3)(1-G_1G_3G_4)(1-G_1G_2G_4)(1-G_2G_3G_4)].$$

By substituting in the appropriate reliability values,

$$\begin{aligned} NVS &= \{1 - [1 - (0.86)(0.79)(0.94)][1 - (0.86)(0.94)(0.68)] \times \\ &\qquad [1 - (0.86)(0.79)(0.68)][1 - (0.79)(0.94)(0.68)]\} \\ &= [1 - (1-0.638636)(1-0.549712)(1-0.461992)(1-0.504968)] \\ &= [1 - (0.361364)(0.450288)(0.538008)(0.495032)] \\ &= 1 - 0.0433368 = 0.9566632 \\ NVS &= 0.95666. \end{aligned}$$

By applying the software reliability model in Section 4.5,

L_1 through $L_n = 0$

$G_1 = (0.95666) \times (0.98) \times (0.93) = 0.8718999$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1$

Therefore,

$$\text{Reliability} = \frac{(0.8718999) \times (1)}{(1)} = 0.8718999$$

$$\text{Reliability} = 0.872.$$

The probability of coincident errors (E) for this example is

$$E = \binom{4}{2}^* (1-G_L)^2 + \binom{4}{3}^* (1-G_L)^3 + \binom{4}{4}^* (1-G_L)^4.$$

The groups for $\binom{4}{2}^*$ are

G_1G_2, G_1G_3, G_1G_4, G_2G_3, G_2G_4, and G_3G_4.

Their respective G_L values are G_1, G_3, G_1, G_3, G_2, and G_3. These values can be grouped as $2 \times G_1 + 2 \times G_2 + 2 \times G_3$.

The groups for $\binom{4}{3}^*$ are $G_1G_2G_3$, $G_1G_2G_4$, $G_1G_3G_4$, and $G_2G_3G_4$, with the G_L values G_3, G_1, G_3, and G_3, respectively. This gives $G_1 + 3 \times G_3$.

The group for $\binom{4}{4}^*$ is $G_1G_2G_3G_4$ with $G_L = G_3$.

Hence,

$$\begin{aligned}
E &= 2 \times (1-G_1)^2 + 2 \times (1-G_2)^2 + 2 \times (1-G_3)^2 + (1-G_1)^3 + \\
&\quad 3 \times (1-G_3)^3 + (1-G_3)^4 \\
&= 2 \times (1-0.86)^2 + 2 \times (1-0.79)^2 + 2 \times (1-0.94)^2 + (1-0.86)^3 \\
&\quad + 3 \times (1-0.94)^3 + (1-0.94)^4 \\
&= 2 \times (0.14)^2 + 2 \times (0.21)^2 + 2 \times (0.06)^2 + (0.14)^3 + 3 \times \\
&\quad (0.06)^3 + (0.06)^4 \\
&= 0.0392 + 0.0882 + 0.0072 + 0.002744 + 0.000648 + 0.00001296 \\
E &= 0.1380049.
\end{aligned}$$

Re-evaluating the reliability value for the N-version software and the overall software reliability value give

$$NVS = 0.9566632 - 0.00001296 = 0.9566503$$
$$NVS = 0.95665$$
$$L_1 \text{ through } L_n = 0$$
$$G_1 = (0.95665) \times (0.98) \times (0.93) = 0.8718908$$
$$G_2 \text{ through } G_K = 0$$
$$\Delta_1 = 1$$
$$\Delta = 1$$

Therefore, the adjusted reliability value is

$$\text{Reliability} = \frac{0.8718908 \times 1}{1} = 0.8718908$$

$$\text{Reliability} = 0.872.$$

Example 3.

This example will be more complicated, involving N-version software in which only x verions are used at a time (reference Figure 3). For this example, the number of versions in the N-version software will be nine. Three of the versions will be run at a time, and their outputs sent to the decision algorithm. If the decision algorithm fails, then the system will rollback, and the next three versions will be run. This cycle will continue until the decision algorithm passes or until all of the versions in the N-version software have been run. Table 7 gives the reliability values for each of the components in this example.

First, the transfer function for the N-version software in which only x versions are used at a time (NVSx) must be determined. The transfer function is a combination of the equations in Section 4.5 for N-version software and a recovery block since the usage of x versions at a time is N-version software, but applying rollback and going through another x versions incorporates the concept of a recovery block. Hence, the transfer function for the N-version software in which only x versions are used at a time is

Software Component	Reliability Value
Version 1	0.84
Version 2	0.71
Version 3	0.66
Version 4	0.87
Version 5	0.92
Version 6	0.90
Version 7	0.73
Version 8	0.85
Version 9	0.78
Decision Algorithm	0.91
Rollback	0.99

Table 7. Reliability Values for the Software Components in the Figure that Represents the N-Version Software in Which Only x Versions are Used at a Time

$$\begin{aligned} NVSx = {} & [1 - (1-G_1G_2)(1-G_1G_3)(1-G_2G_3)] + \\ & \{1-[1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)]\} \times \\ & [1-(1-G_4G_5)(1-G_4G_6)(1-G_5G_6)] + \\ & \{1-[1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)]\} \times \\ & \{1-[1-(1-G_4G_5)(1-G_4G_6)(1-G_5G_6)]\} \times \\ & [1-(1-G_7G_8)(1-G_7G_9)(1-G_8G_9)]. \end{aligned}$$

By substituting in the appropriate reliability values for this example,

$$\begin{aligned} NVSx = {} & \{1-[1-(0.84)(0.71)][1-(0.84)(0.66)][1-(0.71)(0.66)]\} \\ & + [1-\{1-[1-(0.84)(0.71)][1-(0.84)(0.66)][1-(0.71)(0.66)]\}] \\ & \times \{1-[1-(0.87)(0.92)][1-(0.87)(0.90)][1-(0.92)(0.90)]\} \\ & + [1-\{1-[1-(0.84)(0.71)][1-(0.84)(0.66)][1-(0.71)(0.66)]\}] \\ & \times [1-\{1-[1-(0.87)(0.92)][1-(0.87)(0.90)][1-(0.92)(0.90)]\}] \\ & \times \{1-[1-(0.73)(0.85)][1-(0.73)(0.78)][1-(0.85)(0.78)]\} \end{aligned}$$

= [1-(1-0.5964)(1-0.5544)(1-0.4686)] + {1-[1-(1-0.5964) x (1-0.5544)(1-0.4686)]} x [1-(1-0.8004)(1-0.783)(1-0.828)] + {1-[1-(1-0.5964)(1-0.5544)(1-0.4686)]} x {1-[1-(1-0.8004) x (1-0.783)(1-0.828)]} x [1-(1-0.6205)(1-0.5694)(1-0.663)]

= [1-(0.4036)(0.4456)(0.5314)] + {1-[1-(0.4036)(0.4456) x (0.5314)]} x [1-(0.1996)(0.217)(0.172)] + {1-[1-0.4036) x (0.4456)(0.5314)]} x {1-[1-(0.1996)(0.217)(0.172)]}x [1-(0.3795)(0.4306)(0.337)]

= (1-0.095569187) + [1-(1-0.095569187)] x (1-0.0074498704) + [1-(1-0.095569187)] x [1-(1-0.0074498704)] x (1-0.0550701)

= 0.9044309 + (1-0.9044309)(0.9925501) + (1-0.9044309) x (1-0.9925501)(0.9449299)

= 0.9044309 + 0.0948571 + 0.000672771

NVSx = 0.9999608

By applying the software reliability model,

L_1 = (0.9999608) x (0.91) x (1-0.99) = 0.009099643

[The transfer function for rollback is (1.0 - reliability value). (Reference Section 4.5.)]

L_2 through L_n = 0

G_1 = (0.9999608) x (0.91) = 0.9099643

G_2 through G_K = 0

Δ_1 = 1

Δ = 1 - 0.009099643 = 0.9909004

Therefore,

$$\text{Reliability} = \frac{(0.9099643) \times (1)}{(0.9909004)} = 0.9183207$$

Reliability = 0.918.

The accuracy of the hybrid N-version software reliability value will be affected if not all of the n versions are used. (Reference Section 6.1.1.) For this example, it is assumed that only six of the nine versions are actually used. This gives

$$\begin{aligned} NVSx = {} & [1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)] + \\ & \{1-[1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)]\} \times \\ & [1-(1-G_4G_5)(1-G_4G_6)(1-G_5G_6)] \end{aligned}$$

$$= 0.9044309 + 0.0948571$$

$$NVSx = 0.999288$$

$$\begin{aligned} & L_1 = (0.999288) \times (0.91) \times (1-0.99) = 0.009093521 \\ & L_2 \text{ through } L_n = 0 \\ & G_1 = (0.999288) \times (0.91) = 0.90935208 \\ & G_2 \text{ through } G_K = 0 \\ & \Delta_1 = 1 \\ & \Delta = 1-0.0090935 = 0.9909065 \end{aligned}$$

Hence, the overall software reliability value with n = 6 is

$$\text{Reliability} = \frac{(0.90935208) \times (1)}{(0.9909065)} = 0.9176972$$

$$\text{Reliability} = 0.9177 \text{ with } n = 6.$$

It is more difficult to determine the probability of coincident errors for this example. For instance, only the coincident errors within the groups of x versions need to be analyzed. The coincident errors between all of the n versions are irrelevant. Then, since this example combines the equations for N-version software and a recovery block, the probability of coincident errors must be subtracted only from the N-version software.

The probability of errors for the first group of three versions is

$$E_1 = \binom{3}{2}^* (1-G_L)^2 + \binom{3}{3}^* (1-G_L)^3.$$

The groups for $\binom{3}{2}^*$ are G_1G_2, G_1G_3, and G_2G_3, with $G_L = 2 \times G_1 + G_2$. The group for $\binom{3}{3}^*$ is $G_1G_2G_3$ with $G_L = G_1$.

Hence,

$$\begin{aligned} E_1 &= 2 \times (1-G_1)^2 + (1-G_2)^2 + (1-G_1)^3 \\ &= 2 \times (1-0.84)^2 + (1-0.71)^2 + (1-0.84)^3 \\ &= 0.0512 + 0.0841 + 0.004096 \\ E_1 &= 0.139396. \end{aligned}$$

The probability of errors for the second group of three versions is

$$E_2 = \binom{3}{2}^* (1-G_L)^2 + \binom{3}{3}^* (1-G_L)^3.$$

The groups for $\binom{3}{2}^*$ are G_4G_5, G_4G_6, and G_5G_6, with $G_L = 2 \times G_5 + G_6$. The group for $\binom{3}{3}^*$ is $G_4G_5G_6$ with $G_L = G_5$.

The value for E_2 is

$$\begin{aligned} E_2 &= 2 \times (1-G_5)^2 + (1-G_6)^2 + (1-G_5)^3 \\ &= 2 \times (1-0.92)^2 + (1-0.90)^2 + (1-0.92)^3 \\ &= 0.0128 + 0.01 + 0.000512 \\ E_2 &= 0.023312. \end{aligned}$$

The probability of errors for the third group of three versions is

$$E_3 = \binom{3}{2}^* (1-G_L)^2 + \binom{3}{3}^* (1-G_L)^3.$$

The groups of $\binom{3}{2}^*$ for E_3 are G_7G_8, G_7G_9, and G_8G_9, with $G_L = 2 \times G_8 + G_9$. The group of $\binom{3}{3}^*$ is $G_7G_8G_9$ with $G_L = G_8$.

Hence,

$$E_3 = 2 \times (1-0.85)^2 + (1-0.78)^2 + (1-0.85)^3$$
$$= 0.045 + 0.0484 + 0.003375$$
$$E_3 = 0.096775.$$

Considering these probabilities when the transfer function for the N-version software (in which only x versions are used at a time) is calculated gives

$$\begin{aligned} NVSx = {} & [1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)-E_1] + \\ & \{1-[1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)-E_1]\} \times \\ & [1-(1-G_4G_5)(1-G_4G_6)(1-G_5G_6)-E_2] + \\ & \{1-[1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)-E_1]\} \times \\ & \{1-[1-(1-G_4G_5)(1-G_4G_6)(1-G_5G_6)-E_2]\} \times \\ & [1-(1-G_7G_8)(1-G_7G_9)(1-G_8G_9)-E_3] \end{aligned}$$

$$\begin{aligned} = {} & (1-0.095569187 - E_1) + [1-(1-0.095569187 - E_1)] \times \\ & (1-0.0074498704 - E_2) + [1-(1-0.095569187 - E_1)] \times \\ & [1-(1-0.0074498704 - E_2)] \times (1-0.0550701 - E_3) \end{aligned}$$

$$\begin{aligned} = {} & (0.9044309 - E_1) + (1-0.9044309 + E_1)(0.9925501 - E_2) \\ & + (1-0.9044309 + E_1)(1-0.9925501 - E_2)(0.9449299 - E_3) \end{aligned}$$

$$\begin{aligned} = {} & (0.9044309 - 0.139396) + (0.0955691 + 0.139396)(0.9925501 \\ & - 0.023312) + (0.0955691 + 0.139396)(0.0074499 + 0.023312) \\ & \times (0.9449299 - 0.096775) \end{aligned}$$

$$\begin{aligned} = {} & 0.7650349 + (0.2349651)(0.9692381) + (0.2349651)(0.0307619) \\ & \times (0.8481549) \end{aligned}$$

$$= 0.7650349 + 0.2277371 + 0.0061304$$

$$NVSx = 0.9989024.$$

The overall software reliability value with the coincident errors considered is determined as

$L_1 = (0.9989024) \times (0.91) \times (1-0.99) = 0.0090900118$

L_2 through $L_n = 0$

$G_1 = (0.9989024) \times (0.91) = 0.9090012$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1 - 0.0090900118 = 0.99091$

Therefore,

$$\text{Reliability} = \frac{(0.9090012) \times (1)}{(0.99091)} = 0.9173398$$

Reliability = 0.9173.

Example 4.

The hybrid N-version software format, shown in Figure 4, will be evaluated in this example. In this example, the N-version software will consist of three versions. The outputs of these versions are fed into a decision algorithm. If the decision algorithm fails, then the software will rollback and run through the three versions again. However, this time the outputs of the versions are input to an acceptance test prior to entry to the decision algorithm. For this example, it is assumed that the reliability values for the individual software components are as given in Table 8.

The transfer function for the N-version software is

$$\begin{aligned} NVS &= 1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3) \\ &= 1-[1-(0.67)(0.78)][1-(0.67)(0.89)][1-(0.78)(0.89)] \\ &= 1-(1-0.5226)(1-0.5963)(1-0.6942) \\ &= 1-(0.4774)(0.4037)(0.3058) \end{aligned}$$

NVS = 0.94106428.

Software Component	Reliability Value
Version 1	0.67
Version 2	0.78
Version 3	0.89
Acceptance Test	0.86
Decision Algorithm	0.88
Rollback	0.98

Table 8. Reliability Values for the Software Components in the Figure that Represents the N-Version Software in Which the Outputs are Subjected to an Acceptance Test if the Decision Algorithm Fails

The variables of the software reliability model are

$$\text{Closed Loop \#1} = (0.94106428) \times (0.88) \times (1-0.98) = 0.01656273$$

$$\text{Closed Loop \#2} = (0.94106428) \times (1-0.86) \times (0.88) \times (1-0.98) = 0.0023187824$$

[Remember that the transfer function of the equivalent block in a feedback or feed-forward path is (1.0 - reliability value).]

$$\Sigma L_1 = \text{Closed Loop \#1} + \text{Closed Loop \#2} = 0.018881512$$

$$\Sigma L_2 \text{ through } \Sigma L_n = 0$$

$$G_1 = (0.94106428) \times (0.88) = 0.82813657$$

$$G_2 = (0.94106428) \times (1-0.86) \times (0.88) = 0.11593912$$

$$G_3 \text{ through } G_K = 0$$

$$\Delta_1 = 1$$

$$\Delta_2 = 1$$

$$\Delta_3 \text{ through } \Delta_K = 0$$

$$\Delta = 1 - \Sigma L_1 = 1 - 0.018881512 = 0.981118488$$

Therefore,

$$\text{Reliability} = \frac{(0.82813657 \times 1) + (0.11593912 \times 1)}{(0.981118408)} = 0.9622444$$

Reliability = 0.962.

To improve the accuracy of the software reliability model, the probability of coincident errors (E) might be considered. (Reference Section 7.0.) For this example,

$$E = \binom{3}{2}^* (1-G_L)^2 + \binom{3}{3}^* (1-G_L)^3.$$

The groups for $\binom{3}{2}^*$ are G_1G_2, G_1G_3, and G_2G_3 with respective G_L values of G_2, G_3, and G_3, or $G_2 + 2 \times G_3$. The group for $\binom{3}{3}^*$ is $G_1G_2G_3$ with $G_L = G_3$.

Thus,

$$\begin{aligned} E &= (1-G_2)^2 + 2 \times (1-G_3)^2 + (1-G_3)^3 \\ &= (1-0.78)^2 + 2 \times (1-0.89)^2 + (1-0.89)^3 \\ &= (0.22)^2 + 2 \times (0.11)^2 + (0.11)^3 \\ &= 0.0484 + 0.0242 + 0.001331 \\ E &= 0.073931. \end{aligned}$$

By subtracting the probability of coincident errors from the N-version software transfer function, a conservative value of the reliability value for the N-version software and the overall software reliability value can be determined.

$NVS = 0.94106428 - 0.073931 = 0.86713328$

Closed Loop #1 = $(0.86713328) \times (0.88) \times (1-0.98) = 0.015261546$

Closed Loop #2 = $(0.86713328) \times (1-0.86) \times (0.88) \times (1-0.98)$
$= 0.0021366164$

$\Sigma L_1 = 0.015261546 + 0.0021366164 = 0.017398162$

ΣL_2 through $\Sigma L_n = 0$

$G_1 = (0.86713328) \times (0.88) = 0.76307729$

$G_2 = (0.86713328) \times (1-0.86) \times (0.88) = 0.10683082$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - \Sigma L_1 = 1 - 0.017398162 = 0.98260184$

Therefore,

$$\text{Reliability} = \frac{(0.76307729 \times 1) + (0.10683082 \times 1)}{(0.98260184)} = 0.88531089$$

Reliability = 0.885.

APPENDIX II
RECOVERY BLOCK CALCULATIONS

The transfer function for a recovery block is dependent upon the number of alternates (n) that are used. This transfer function is calculated with the following equation:

$G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + \ldots.$
with G_i = the reliability value for alternate i and
$i = 1, 2, 3, \ldots n.$

The examples below demonstrate the determination of the overall software reliability value with this equation and the software reliability model.

Example 1

Figure 5 shows the general format of a backward recovery block. For this example, the number of alternates will be four. The reliability value for each of the software components is listed in Table 9.

Software Component	Reliability Value
Alternate 1	0.86
Alternate 2	0.75
Alternate 3	0.79
Alternate 4	0.84
Acceptance Test	0.91
Rollback	0.93

Table 9. Reliability Values for the Software Components in the Figure that Represents the General Format of a Backward Recovery Block

The transfer function for the recovery block (RB) is

$$RB = G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + (1 - G_1)(1 - G_2)(1 - G_3)G_4$$

$$= 0.86 + (1 - 0.86)(0.75) + (1 - 0.86)(1 - 0.75)(0.79) + (1 - 0.86)(1 - 0.75)(1 - 0.79)(0.84)$$

$$= 0.86 + (0.14)(0.75) + (0.14)(0.25)(0.79) + (0.14)(0.25)(0.21)(0.84)$$

$$= 0.86 + 0.105 + 0.02765 + 0.006174$$

$$RB = 0.998824$$

The variables of the software reliability model will be

$$L_1 = (0.998824) \times (0.91) \times (1.0 - 0.93) = 0.0636251$$

[Recall that the transfer function ur rollback is (1.0 - reliability value). This was defined as such in Section 4.5.]

L_2 through $L_n = 0$
$G_1 = (0.998824) \times (0.91) = 0.9089298$
G_2 through $G_K = 0$
$\Delta_1 = 1$
Δ_2 through $\Delta_K = 0$
$\Delta = 1 - 0.0636251 = 0.9363749$

Therefore,

$$\text{Reliability} = \frac{(0.9089298) \times (1)}{(0.9363749)} = 0.9706901$$

Reliability = 0.97.

As was discussed in Section 6.1.2, if only two of the alternates are actually used, although the recovery block supplies four alternates, this will decrease the reliability of the recovery block and consequently decrease the overall software reliability. The following calculations show this.

$$RB = G_1 + (1 - G_1)G_2 = 0.86 + (1 - 0.86)(0.75)$$
$$= 0.86 + 0.105 = 0.965$$

$L_1 = (0.965) \times (0.91) \times (1 - 0.93) = 0.0614705$

L_2 through $L_n = 0$

$G_1 = (0.965) \times (0.91) = 0.9089298$

G_2 through $G_K = 0$

$\Delta_1 = 1$

Δ_2 through $\Delta_K = 0$

$\Delta = 1 - 0.0614705 = 0.9385295$

Hence,

$$\text{Reliability} = \frac{(0.9089298) \times (1)}{(0.9385295)} = 0.9684616$$

Reliability = 0.968 when only two of the alternates are used.

Example 2

The general format of a forward recovery block is shown in Figure 6. This example will evaluate the overall software reliability of this figure (six alternates will be used: one primary alternate and five additional alternates), with the reliability values assigned as shown in Table 10.

The transfer function for the recovery block (RB) is

$$\begin{aligned} RB = {} & G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + \\ & (1 - G_1)(1 - G_2)(1 - G_3)G_4 + \\ & (1 - G_1)(1 - G_2)(1 - G_3)(1 - G_4)G_5 + \\ & (1 - G_1)(1 - G_2)(1 - G_3)(1 - G_4)(1 - G_5)G_6 \end{aligned}$$

Software Component	Reliability Value
Alternate 1	0.81
Alternate 2	0.72
Alternate 3	0.73
Alternate 4	0.74
Alternate 5	0.85
Alternate 6	0.86
Acceptance Test	0.97
Rollback	0.98
Roll-Forward	0.89

Table 10. Reliability Values for the Software Components in the Figure that Represents the General Format of a Forward Recovery Block

$$= 0.81 + (1 - 0.81)(0.72) + (1 - 0.81)(1 - 0.72)(0.73) + (1 - 0.81)(1 - 0.72)(1 - 0.73)(0.74) + (1 - 0.81)(1 - 0.72)(1 - 0.73)(1 - 0.74)(0.85) + (1 - 0.81)(1 - 0.72)(1 - 0.73)(1 - 0.74)(1 - 0.85)(0.86)$$

$$= 0.81 + (0.19)(0.72) + (0.19)(0.28)(0.73) + (0.19)(0.28)(0.27)(0.74) + (0.19)(0.28)(0.27)(0.26)(0.85) + (0.19)(0.28)(0.27)(0.26)(0.15)(0.86)$$

$$= 0.81 + 0.1368 + 0.038836 + 0.01062936 + 0.003174444 + 0.00048176856$$

$$= 0.99992157256$$

$$RB = 0.9999216.$$

With the software reliability model,

$$L_1 = (0.9999216) \times (0.97) \times (1 - 0.98) = 0.019398479$$

[Note that the transfer function for rollback is (1.0 - reliability value).]

L_2 through $L_n = 0$

$G_1 = (0.9999216) \times (0.97) = 0.96992395$

$G_2 = (0.9999216) \times (0.97) \times (1 - 0.98) \times (1 - 0.89) = 0.0021338327$

[Remember that the transfer function for rollback and roll-forward are (1.0 - reliability value). This was discussed in Section 4.5.]

G_3 through $G_K = 0$
$\Delta_1 = 1$
$\Delta_2 = 1$
Δ_3 through $\Delta_K = 0$
$\Delta = 1 - 0.019398479 = 0.98060152$

Therefore,

$$\text{Reliability} = \frac{(0.96992395 \times 1) + (0.0021338327 \times 1)}{(0.98060152)}$$

$$= (0.97205778)/(0.98060152) = 0.99128725$$

Reliability = 0.991.

Discussion of the Results:

This result is as expected. With just the n alternates, acceptance test, and rollback, the overall reliability would be

$$\text{Reliability} = \frac{(0.9999216)(0.97)}{1 - (0.9999216)(0.97)(1 - 0.98)}$$

Reliability = 0.9891112 = 0.989.

The roll-forward should increase this reliability value, as it does.

To evaluate the effect on accuracy when less than the n alternates (in this example n = 6) are actually used, the reliability of this example will be evaluated with n = 3, n = 4, and n = 5.

For n = 3,

$$RB = G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3$$
$$= 0.81 + (1 - 0.81)(0.72) + (1 - 0.81)(1 - 0.72)(0.73)$$
$$= 0.81 + 0.1368 + 0.038836$$
$$RB = 0.985636$$

Using the software reliability model,

$L_1 = (0.985636)(0.97)(1 - 0.98) = 0.019121338$

L_2 through $L_n = 0$

$G_1 = (0.985636)(0.97) = 0.95606692$

$G_2 = (0.985636)(0.97)(1 - 0.98)(1 - 0.89) = 0.0021033472$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - 0.019121338 = 0.98087866$

gives $$\text{Reliability} = \frac{(0.95606692 \times 1) + (0.0021033472 \times 1)}{(0.98087866)} = 0.97684893$$

Reliability = 0.977.

For $n = 4$,

$RB = 0.81 + 0.1368 + 0.038836 + 0.01062936$

$RB = 0.99626536$

Using the software reliability model,

$L_1 = (0.99626536)(0.97)(1 - 0.98) = 0.019327547$

L_2 through $L_n = 0$

$G_1 = (0.99626536)(0.97) = 0.9663774$

$G_2 = (0.99626536)(0.97)(1 - 0.98)(1 - 0.89) = 0.0021260303$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - 0.019327547 = 0.98067245$

gives $$\text{Reliability} = \frac{(0.9663774 \times 1) + (0.0021260303 \times 1)}{(0.98067245)} = 0.98759115$$

Reliability = 0.988.

For n = 5,

$$RB = 0.99626536 + 0.003174444$$

$$RB = 0.999439804$$

Using the software reliability model,

$$L_1 = (0.999439804)(0.97)(1 - 0.98) = 0.019389132$$

$$L_2 \text{ through } L_n = 0$$

$$G_1 = (0.999439804)(0.97) = 0.96945661$$

$$G_2 = (0.999439804)(0.97)(1 - 0.98)(1 - 0.89) = 0.0021328045$$

$$G_3 \text{ through } G_K = 0$$

$$\Delta_1 = 1$$

$$\Delta_2 = 1$$

$$\Delta_3 \text{ through } \Delta_K = 0$$

$$\Delta = 1 - 0.019389132 = 0.98061087$$

gives $$\text{Reliability} = \frac{(0.96945661 \times 1) + (0.0021328045 \times 1)}{(0.98061087)} = 0.99080017$$

Reliability = 0.991.

The following table compares the reliability values that are obtained by using less than n alternates in this example.

Number of Alternates Used	Recovery Block Reliability Value	Overall Software Reliability Value
n = 3	0.98564	0.977
n = 4	0.99627	0.988
n = 5	0.99944	0.991
n = 6	0.99992	0.991

Table 11. Accuracy Effects on This Example When Less Than n Alternates are Actually used

Discussion of the Results:

This is as expected. As stated in Section 6.1.2, by actually using fewer than the n alternates, the reliability values for the recovery block and the overall software will generally decrease. However, as was seen in the case with n = 5, by not using the sixth alternate (which has a reliability value of 0.86 in this example), an extremely slight increase in reliability was found.

Example 3

Figure 7 shows a possible variation of a forward recovery block. For this example, the number of alternates will be three. The reliability value for each of the software components is listed in Table 12.

Software Component	Reliability Value
Alternate 1	0.80
Alternate 2	0.70
Alternate 3	0.90
Acceptance Test	0.98
Any Process	0.97
Rollback	0.95
Roll-Forward	0.96

Table 12. Reliability Values for the Software Components in the Figure that Represents a Variation of the Forward Recovery Block

The transfer function for the recovery block (RB) is

$$\begin{aligned} RB &= G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 \\ &= 0.80 + (1 - 0.80) \times (0.70) + (1 - 0.80) \times (1 - 0.70) \times (0.90) \\ &= 0.80 + (0.20 \times 0.70) + (0.20 \times 0.30 \times 0.90) \\ &= 0.80 + 0.14 + 0.054 \\ RB &= 0.994. \end{aligned}$$

The variables of the software reliability model are

$L_1 = (0.994) \times (0.98) \times (1 - 0.95) = 0.048706$

L_2 through $L_n = 0$

$G_1 = (0.994) \times (0.98) \times (0.97) = 0.9448964$

$G_2 = (0.994) \times (0.98) \times (1 - 0.95) \times (1 - 0.96) = 0.0019482$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - L_1 = 1 - 0.048706 = 0.951294$

Therefore,

$$\text{Reliability} = \frac{(0.9448964 \times 1) + (0.0019482 \times 1)}{(0.951294)} = 0.99532279$$

Reliability = 0.995.

To demonstrate the effect on accuracy if less than the n alternates (in this example n = 3) are actually used, the reliability of the recovery block and overall software reliability value will be re-calculated for n = 1 and n = 2.

For n = 1,

RB = 0.80.

With the software reliability model,

$L_1 = (0.80) \times (0.98) \times (1 - 0.95) = 0.0392$

L_2 through $L_n = 0$

$G_1 = (0.80) \times (0.98) \times (0.97) = 0.76048$

$G_2 = (0.80) \times (0.98) \times (1 - 0.95) \times (1 - 0.96) = 0.001568$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - L_1 = 1 - 0.0392 = 0.9608$

$$\text{Reliability} = \frac{(0.76048 \times 1) + (0.001568 \times 1)}{(0.9608)} = 0.7931391$$

Reliability = 0.793.

For n = 2,

RB = 0.80 + 0.14 = 0.94.

With the software reliability model,

$L_1 = (0.94) \times (0.98) \times (1 - 0.95) = 0.04606$

L_2 through $L_n = 0$

$G_1 = (0.94) \times (0.98) \times (0.97) = 0.893564$

$G_2 = (0.94) \times (0.98) \times (1 - 0.95) \times (1 - 0.96) = 0.0018424$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - L_i = 1 - 0.04606 = 0.95394$

$$\text{Reliability} = \frac{(0.893564 \times 1) + (0.0018424 \times 1)}{(0.95394)} = 0.93864017$$

Reliability = 0.939.

Table 13 compares the reliability values of the recovery block and the overall software reliability values that are obtained by using all or less than the n alternates in this example.

Number of Alternates Used	Recovery Block Reliability Value	Overall Software Reliability Value
n = 1	0.80	0.793
n = 2	0.94	0.939
n = 3	0.994	0.995

Table 13. Comparison of Reliability Values When Less Than n Alternates are Actually Used

(This page left blank intentionally)

APPENDIX III
FEEDBACK LOOP CALCULATIONS

For basic feedback loops such as those shown in Figures 3, 4, and 5, the ideal software reliability value of the individual blocks and overall software reliability is 1.0. With an original block diagram of the form

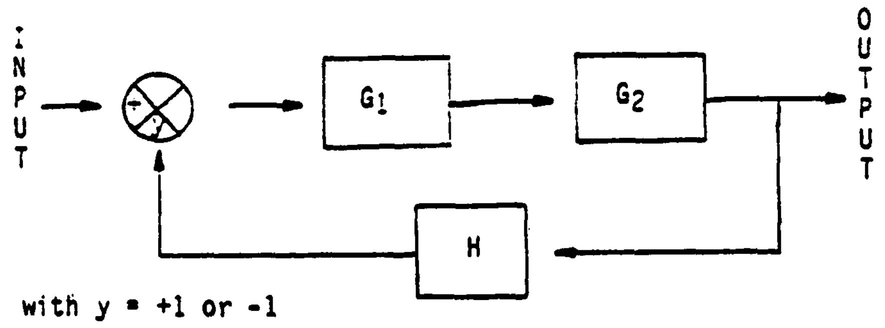


Figure 12. Basic Feedback Loop

the block diagram transformation to eliminate a feedback loop gives the equivalent block diagram of the form

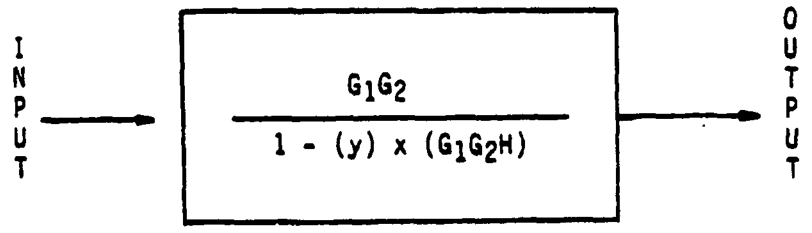


Figure 13. Basic Feedback Loop Equivalent

If it is a negative feedback loop, the equivalent transfer function is:

$$\frac{G_1G_2}{1 + G_1G_2H}$$

With $G_1 = 1.0$, $G_2 = 1.0$, and $H = 1.0$, the overall reliability value would be 0.5, which is undesirable. It is desired that the overall software rel'ability and individual block reliability values be 1.0. With the negative feedback loop and these goals, there are three cases to be evaluated.

Negative Feedback Loop, Case 1:

Want: $1 = \dfrac{G_1G_2}{1 + G_1G_2H}$

If: $G_1 = 1.0$ and $H = 1.0$

Then: $\dfrac{G_2}{1 + G_2} = 1.0$ or $1 + G_2 = G_2$.

Conclusion: This is invalid.

Negative Feedback Loop, Case 2:

Want: $1.0 = \dfrac{G_1G_2}{1 + G_1G_2H}$

If: $G_2 = 1.0$ and $H = 1.0$

Then: $1 + G_1 = G_1$

Conclusion: This is invalid.

Negative Feedback Loop, Case 3:

$$\text{Want:} \quad 1.0 = \frac{G_1 G_2}{1 + G_1 G_2 H}$$

If: $G_1 = 1.0$ and $G_2 = 1.0$

$$\text{Then:} \quad \frac{1}{1 + H} = 1.0 \quad \text{or} \quad 1 + H = 1$$

Conclusion: $H = 0$.

If it is a positive feedback loop, the equivalent transfer function is:

$$\frac{G_1 G_2}{1 - G_1 G_2 H}$$

As $G_1 \Rightarrow 1.0$, $G_2 \Rightarrow 1.0$, and $H \Rightarrow 1.0$, the overall reliability value approaches $+\infty$. Again, it is desired that the overall software reliability and individual block reliability values be 1.0. There are three cases to be evaluated with the positive feedback loop.

Positive Feedback Loop, Case 1:

$$\text{Want:} \quad 1.0 = \frac{G_1 G_2}{1 - G_1 G_2 H}$$

If: $G_1 = 1.0$ and $H = 1.0$

$$\text{Then:} \quad \frac{G_2}{1 - G_2} = 1.0 \quad \text{or} \quad 1 - G_2 = G_2 \quad \text{or} \quad G_2 = 0.5$$

Conclusion: This is undesirable.

Positive Feedback Loop, Case 2:

Want: $1.0 = \dfrac{G_1G_2}{1 - G_1G_2H}$

If: $G_2 = 1.0$ and $H = 1.0$

Then: $\dfrac{G_1}{1 - G_1} = 1.0$ or $1 - G_1 = G_1$ or $G = 0.5$

Conclusion: This is undesirable.

Positive Feedback Loop, Case 3:

Want: $1.0 = \dfrac{G_1G_2}{1 - G_1G_2H}$

If: $G_1 = 1.0$ and $G_2 = 1.0$

Then: $\dfrac{1}{1 - H} = 1.0$ or $1 - H = 1$

Conclusion: $H = 0$.

By comparing the results of the positive and negative feedback cases (since it is desired that the software reliability model should accommodate both of these options), it is obvious that the transfer function of H must equal zero. Therefore, the transfer function of the equivalent block in any feedback path is (1.0 - reliability value).

APPENDIX IV
FEED-FORWARD CALCULATIONS

Figures 6 and 7 are examples of block diagrams involving feed-forward paths. In an ideal situation, the reliability value of the individual blocks (G_1, G_2, G_3,...G_n) and the overall software reliability (R) are 1.0. It is important to remember that reliability is defined such that $0 \leq G_1$, G_2, G_3,...G_n, $R \leq 1.0$. Figure 14 shows the block diagram for a basic feed-forward path.

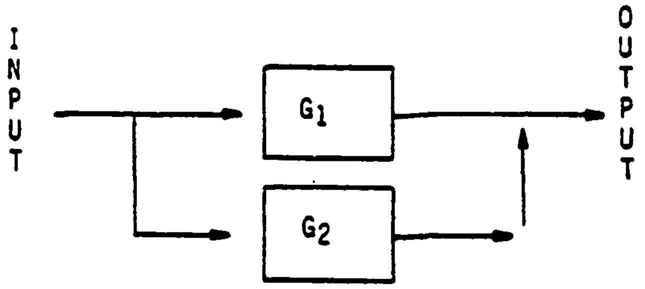


Figure 14. Basic Feed-Forward Path

A block diagram transformation to eliminate the feed-forward loop gives the following equivalent block diagram.

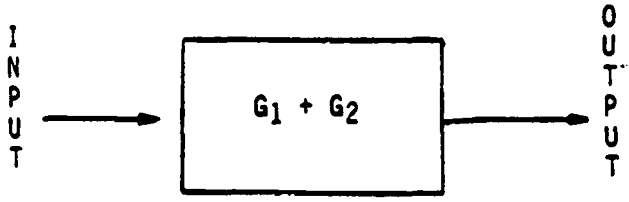


Figure 15. Basic Feed-Forward Path Equivalent

Ideally, if the reliability value of the individual blocks is 1.0, then the overall reliability should be 1.0. Therefore,

Want: $1.0 = G_1 + G_2$

If: $G_1 = 1.0$

Then: $G_2 = 0$ or $1.0 = 1.0 + (1.0 - G_2)$

Conclusion: The transfer function of the equivalent block in any feed-forward path is (1.0 - reliability value).

APPENDIX V
ANALYSIS OF SCOTT'S RECOVERY BLOCK RELIABILITY MODEL

In Scott's recovery block reliability model, the variables are defined as:

$P(C_i)$ = the probability of alternate i executing correctly;

$P(I_i) = 1 - P(C_i)$;

$P(C_R)$ = the probability of the recovery program executing correctly;

$P(I_R) = 1 - P(C_R)$;

$P(A_I)$ = the probability of accepting an incorrect result;

$P(R_I)$ = the probability of rejecting an incorrect result
$= 1 - P(A_I)$;

$P(R_C)$ = the probability of rejecting a correct result;

$P(A_C)$ = the probability of accepting a correct result
$= 1 - P(R_C)$;

Type 1 Error = the program alternate produces an incorrect result, but the acceptance test labels the result as correct;

Type 2 Error = the final alternate produces correct results, but the acceptance test erroneously determines that the results are incorrect;

Type 3 Error = the recovery program cannot successfully recover the input state of the previous alternate in preparation for executing another alternate or could not successfully invoke the next alternate;

Type 4 Error = the last alternate produces incorrect results and the acceptance test judges that the results are incorrect;

n = the number of alternates; and

R = the reliability.

The results of Scott's recovery block reliability model will be compared to those that would be obtained in the software reliability model that has been proposed. The block diagram for the recovery block that is described by Scott would look like:

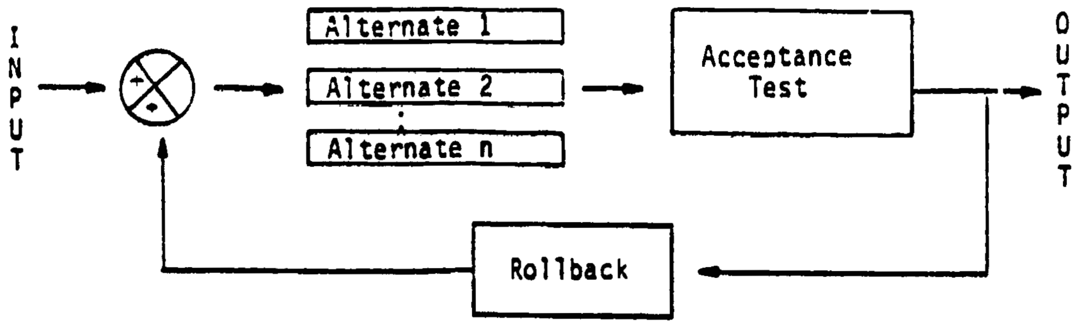


Figure 16. Basic Recovery Block

For n = 1, the feedback loop would be deleted, giving a block diagram of

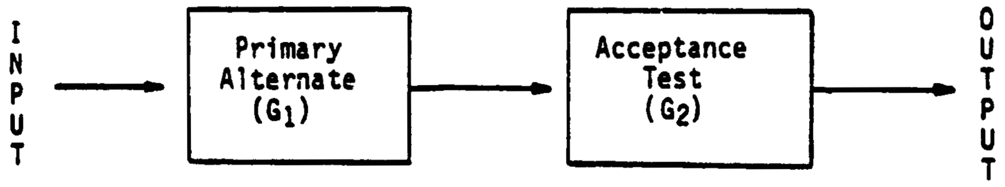


Figure 17. Special Case Recovery Block with Only One Alternate

with an overall transfer function of $R = G_1 \times G_2$. This special case with $n = 1$ is computed with Scott's recovery block reliability model as

$$R = 1 - [\text{Type 1 Error} + \text{Type 2 Error} + \text{Type 3 Error} + \text{Type 4 Error}],$$

with Type 1 Error $= P(I_1)P(A_I)$;
Type 2 Error $= P(C_1)P(R_C)$;
Type 3 Error $= 0$; and
Type 4 Error $= P(I_1)P(R_I)$.

By substitution,

$$G_1G_2 = 1 - [P(I_1)P(A_I) + P(C_1)P(R_C) + P(I_1)P(R_I)].$$

By further substitution,

$$G_1G_2 = 1 - \{[1 - P(C_1)][1 - P(R_I)] + P(C_1)[1-P(A_C)] + [1 - P(C_1)]P(R_I)\}.$$

Multiplying out the factors gives

$$G_1G_2 = 1 - \{1 + P(C_1)P(R_I) - P(C_1) - P(R_I) + P(C_1) - P(C_1)P(A_C) + P(R_I) - P(C_1)P(R_I)\}.$$

This can be reduced to

$$G_1G_2 = 1 - [1 - P(C_1)P(A_C)] = P(C_1)P(A_C).$$

This is as expected, with $G_1 = P(C_1)$ and $G_2 = P(A_C)$.

For n = 2,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [\text{Type 1 Error} + \text{Type 2 Error} + \text{Type 3 Error} + \text{Type 4 Error}].$$

By substitution,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [\{P(I_1)P(A_I) + [P(I_1)P(A_I) - 0] \times [\frac{P(C_R)P(I_2)}{P(I_1)}] \times [P(C_1)P(R_C) + P(I_1)P(R_I)]\} + \{P(C_1)P(R_C)P(C_R)P(C_2) \times [P(R_C) + \frac{P(I_1)P(R_I)}{P(C_1)}]\} + \{P(C_1)P(R_C)P(I_R) + P(I_1)P(R_I)P(I_R)\} + \{P(I_1)P(R_I)P(C_R)P(I_2)[P(R_I) + \frac{P(C_1)P(R_C)}{P(I_1)}]\}].$$

Multiplying out the factors and cancelling alike numerator and denominator terms gives

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [P(I_1)P(A_I) + P(A_I)P(C_R)P(I_2)P(C_1)P(R_C) + P(A_I)P(C_R)P(I_2)P(I_1)P(R_I) + P(C_1)P(R_C)P(C_R)P(C_2)P(R_C) + P(R_C)P(C_R)P(C_2)P(I_1)P(R_I) + P(C_1)P(R_C)P(I_R) + P(I_1)P(R_I)P(I_R) + P(I_1)P(R_I)P(C_R)P(I_2)P(R_I) + P(R_I)P(C_R)P(I_2)P(C_1)P(R_C)].$$

By substitution,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - \{[1 - P(C_1)][1 - P(R_I)] + [1 - P(R_I)]P(C_R)[1 - P(C_2)]P(C_1) \times [1 - P(A_C)] + [1 - P(R_I)]P(C_R) \times [1 - P(C_2)][1 - P(C_1)]P(R_I) + P(C_1)[1 - P(A_C)]P(C_R)P(C_2)[1 - P(A_C)] + [1 - P(A_C)]P(C_R)P(C_2)[1 - P(C_1)]P(R_I) + P(C_1)[1 - P(A_C)][1 - P(C_R)] + [1 - P(C_1)]P(R_I)[1 - P(C_R)] + [1 - P(C_1)]P(R_I)P(C_R)[1 - P(C_2)]P(R_I) + P(R_I)P(C_R)[1 - P(C_2)]P(C_1)[1 - P(A_C)]\}.$$

With the factors multiplied out,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [1 + P(C_1)P(R_I) - P(C_1) - P(R_I) + P(C_1)P(C_R) - P(C_1)P(C_2)P(C_R) - P(C_1)P(C_R)P(R_I) + P(C_1)P(C_2)P(C_R)P(R_I) - P(A_C)P(C_1)P(C_R) + P(A_C)P(C_1)P(C_2)P(C_R) + P(A_C)P(C_1)P(C_R)P(R_I) - P(A_C)P(C_1)P(C_2)P(C_R)P(R_I) + P(C_R)P(R_I) - P(C_1)P(C_R)F(R_I) - P(C_2)P(C_R)P(R_I) + P(C_1)P(C_2)F(C_R)P(R_I) - P(C_R)P(R_I)P(R_I) + P(C_1)P(C_R)P(R_I)P(R_I) + P(C_2)P(C_R)P(R_I)P(R_I) - P(C_1)P(C_2)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_2)P(C_R) -$$

$P(A_C)P(C_1)P(C_2)P(C_R) - P(A_C)P(C_1)P(C_2)P(C_R) +$
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_R) + P(C_2)P(C_R)P(R_I) -$
$P(C_1)P(C_2)P(C_R)P(R_I) - P(A_C)P(C_2)P(C_R)P(R_I) +$
$P(A_C)P(C_1)P(C_2)P(C_R)P(R_I) + P(C_1) - P(C_1)P(C_R) -$
$P(A_C)P(C_1) + P(A_C)P(C_1)P(C_R) + P(R_I) -$
$P(C_R)P(R_I) - P(C_1)P(R_I) + P(C_1)P(C_R)P(R_I) +$
$P(C_R)P(R_I)P(R_I) - P(C_2)P(C_R)P(R_I)P(R_I) -$
$P(C_1)P(C_R)P(R_I)P(R_I) +$
$P(C_1)P(C_2)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_R)P(R_I) -$
$P(A_C)P(C_1)P(C_R)P(R_I) - P(C_1)P(C_2)P(C_R)P(R_I) +$
$P(A_C)P(C_1)P(C_2)P(C_R)P(R_I)].$

By cancellation,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [1 - P(A_C)P(C_1)P(C_2)P(C_R) + P(A_C)P(A_C)P(C_1)P(C_2)P(C_R) - P(A_C)P(C_2)P(C_R)P(R_I) - P(A_C)P(C_1) + P(A_C)P(C_1)P(C_2)P(C_R)P(R_I)].$$

Rearranging gives

$$\frac{G_1G_2}{1 - G_1G_2H} = P(A_C)[P(C_1)P(C_2)P(C_R) - P(A_C)P(C_1)P(C_2)P(C_R) + P(C_2)P(C_R)P(R_I) + P(C_1) - P(C_1)P(C_2)P(C_R)P(R_I)].$$

For n = 3,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [\text{Type 1 Error} + \text{Type 2 Error} + \text{Type 3 Error} + \text{Type 4 Error}].$$

By substitution,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [\{P(I_1)P(A_I) + P(A_I)P(C_R)P(I_2)[P(C_1)P(R_C) + P(I_1)P(R_I)] + P(A_I)P(C_R)P(I_2)[P(C_1)P(R_C) + P(I_1)P(R_I)] \times$$

$$\left[\frac{P(C_R)P(I_3)}{P(I_2)}\right][P(C_2)P(R_C) + P(I_2)P(R_I)]\} +$$
$$\{P(C_1)P(C_2)P(C_R)P(R_C)P(R_C)P(C_R)P(C_3) \times$$
$$\left[P(R_C) + \frac{P(I_2)P(R_I)}{P(C_2)}\right] + P(C_2)P(C_R)P(I_1) \times$$
$$P(R_C)P(R_I)P(C_R)P(C_3)\left[P(R_C) + \frac{P(I_2)P(R_I)}{P(C_2)}\right]\} +$$
$$\{P(C_1)P(R_C)P(I_R) + P(I_1)P(R_I)P(I_R) +$$
$$[P(C_1)P(R_C)P(I_R) + P(I_1)P(R_I)P(I_R)] \times$$
$$P(C_R)[P(C_2)P(R_C) + P(I_3)P(R_I)]\} +$$
$$\{[P(I_1)P(R_I)P(C_R)P(I_2)\left[P(R_I) + \frac{P(C_1)P(R_C)}{P(I_1)}\right] \times$$
$$P(C_R)P(I_3)\left[P(R_I) + \frac{P(C_2)P(R_C)}{P(I_2)}\right]\}].$$

By multiplying out the terms (and arranging the coefficients in alphabetical and numerical order),

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - \{P(A_I)P(I_1) + P(A_I)P(C_1)P(C_R)P(I_2)P(R_C) +$$
$$P(A_I)P(C_R)P(I_1)P(I_2)P(R_I) +$$
$$[P(A_I)P(C_1)P(C_R)P(C_R)P(I_3)P(R_C) +$$
$$P(A_I)P(C_R)P(C_R)P(I_1)P(I_3)P(R_I)] \times$$
$$[P(C_2)P(R_C) + P(I_2)P(R_I)] +$$
$$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_C)P(R_C)P(R_C) +$$
$$P(C_1)P(C_3)P(C_R)P(C_R)P(I_2)P(R_C)P(R_C)P(R_I) +$$
$$P(C_2)P(C_3)P(C_R)P(C_R)P(I_1)P(R_C)P(R_C)P(R_I) +$$
$$P(C_3)P(C_R)P(C_R)P(I_1)P(I_2)P(R_C)P(R_I)P(R_I) +$$
$$P(C_1)P(I_R)P(R_C) + P(I_1)P(I_R)P(R_I) +$$
$$P(C_1)P(C_2)P(C_R)P(I_R)P(R_C)P(R_C) +$$
$$P(C_1)P(C_R)P(I_3)P(I_R)P(R_C)P(R_I) +$$
$$P(C_2)P(C_R)P(I_1)P(I_R)P(R_C)P(R_I) +$$
$$P(C_R)P(I_1)P(I_3)P(I_R)P(R_I)P(R_I) +$$
$$[P(C_R)P(C_R)P(I_1)P(I_2)P(I_3)P(R_I)P(R_I) +$$
$$P(C_1)P(C_R)P(C_R)P(I_2)P(I_3)P(R_C)P(R_I)] \times$$
$$\left[P(R_I) + \frac{P(C_2)P(R_C)}{P(I_2)}\right]\}.$$

By substitution [to eliminate the $P(A_I)$ and $P(I_i)$ terms],

$$
\begin{aligned}
\frac{G_1G_2}{1 - G_1G_2H} = 1 - \Big[& [1-P(C_1)][1 - P(R_I)] + \\
& [1 - P(R_I)]P(C_1)P(C_R)][1 - P(C_2)][1 - P(A_C)] + \\
& [1 - P(R_I)]P(C_R)[1 - P(C_1)][1 - P(C_2)]P(R_1) + \\
& \{[1 - P(R_I)][P(C_1)P(C_R)P(C_R)[1 - P(C_3)][1 - P(A_C)] + \\
& \quad [1 - P(R_I)]P(C_R)P(C_R)[1 - P(C_1)][1 - P(C_3)]P(R_I)\} \times \\
& \quad \{P(C_2)[1 - P(A_C)] + [1 - P(C_2)]P(R_I)\} + \\
& P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)[1 - P(A_C)][1 - P(A_C)][1 - P(A_C)] + \\
& P(C_1)P(C_3)P(C_R)P(C_R)[1 - P(C_2)][1 - P(A_C)][1 - P(A_C)]P(R_I) + \\
& P(C_2)P(C_3)P(C_R)P(C_R)[1 - P(C_1)][1 - P(A_C)][1 - P(A_C)] \times \\
& P(R_I) + P(C_3)P(C_R)P(C_R)[1 - P(C_1)][1 - P(C_2)][1 - P(A_C)] \times \\
& P(R_I)P(R_I) + P(C_1)[1 - P(C_R)][1 - P(A_C)] + [1 - P(C_1)] \times \\
& [1 - P(C_R)]P(R_I) + P(C_1)P(C_2)P(C_R)[1 - P(C_R)][1 - P(A_C)] \times \\
& [1 - P(A_C)] + P(C_1)P(C_R)[1 - P(C_3)][1 - P(C_R)][1 - P(A_C)] \times \\
& P(R_I) + P(C_2)P(C_R)[1 - P(C_1)][1 - P(C_R)][1 - P(A_C)] \times \\
& P(R_I) + P(C_R)[1 - P(C_1)][1 - P(C_3)][1 - P(C_R)]P(R_I) \times \\
& P(R_I) + P(C_R)P(C_R)[1 - P(C_1)][1 - P(C_2)][1 - P(C_3)] \times \\
& P(R_I)P(R_I) + P(C_1)P(C_R)P(C_R)[1 - P(C_2)][1 - P(C_3)] \times \\
& [1 - P(A_C)]P(R_I) \times \left\{P(R_I) + \frac{P(C_2)[1 - P(A_C)]}{[1 - P(C_2)]}\right\}\Big]
\end{aligned}
$$

Multiplying the terms out gives:

$$
\begin{aligned}
\frac{G_1G_2}{1 - G_1G_2H} = 1 - \; & [1 - P(C_1) - P(R_I) + P(C_1)P(R_I) + P(C_1)P(C_R) - \\
& P(C_1)P(C_2)P(C_R) - P(A_C)P(C_1)P(C_R) + P(A_C)P(C_1)P(C_2)P(C_R) - \\
& P(C_1)P(C_R)P(R_I) + P(C_1)P(C_2)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_R)P(R_I) - \\
& P(A_C)P(C_1)P(C_2)P(C_R)P(R_I) + P(C_R)P(R_I) - P(C_1)P(C_R)P(R_I) - \\
& P(C_2)P(C_R)P(R_I) + P(C_1)P(C_2)P(C_R)P(R_I) - \\
& P(C_R)P(R_I)P(R_I) + P(C_1)P(C_R)P(R_I)P(R_I) + P(C_2)P(C_R)P(R_I)P(R_I) - \\
& P(C_1)P(C_2)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_2)P(C_R)P(C_R) - \\
& P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) - P(A_C)P(C_1)P(C_2)P(C_R)P(C_R) + \\
& P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) - P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) + \\
& P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) -
\end{aligned}
$$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) + P(C_2)P(C_R)P(C_R)P(R_I) -$

$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) - P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) +$

$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) - P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) + P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) - P(A_C)P(C_1)P(C_2)P(C_R)P(C_R) +$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) + P(A_C)P(A_C)P(C_1)P(C_2)P(C_R)P(C_R) -$

$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) +$

$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) -$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$

$P(A_C)P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$

$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$

$P(A_C)P(C_2)P(C_R)P(C_R)P(R_I) +$

$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$

$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) +$

$P(A_C)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(C_1)P(C_R)P(C_R)P(R_I) - P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) -$

$P(A_C)P(C_1)P(C_R)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) -$

$P(C_1)P(C_R)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(A_C)P(C_1)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(C_R)P(C_R)P(R_I)P(R_I) - P(C_1)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) +$

$P(C_1)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) + P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) -$

$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) - P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$

$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) -$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) +$

$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$

$P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ -
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ +
$P(C_2)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I)$ -
$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I)$ -
$P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I)$ +
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I)$ +
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ -
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ -
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ -
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ +
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ +
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ +
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ -
$P(A_C)P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)$ +
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)$ - $P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)$ -
$P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(A_C)P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)$ -
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ -
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ - $P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ -
$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(A_C)P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ -
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ -
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)$ +
$P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ -
$P(A_C)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ - $P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ +
$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ -
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ +
$P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ +
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)$ -

$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) + P(C_1) -$
$P(A_C)P(C_1) - P(C_1)P(C_R) + P(A_C)P(C_1)P(C_R) + P(R_I) -$
$P(C_R)P(R_I) - P(C_1)P(R_I) + P(C_1)P(C_R)P(R_I) + P(C_1)P(C_2)P(C_R) -$
$P(C_1)P(C_2)P(C_R)P(C_R) - P(A_C)P(C_1)P(C_2)P(C_R) -$
$P(A_C)P(C_1)P(C_2)P(C_R) + P(A_C)P(C_1)P(C_2)P(C_R)P(C_R) +$
$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R) - P(A_C)P(A_C)P(C_1)P(C_2)P(C_R)P(C_R) +$
$P(C_1)P(C_R)P(R_I) - P(C_1)P(C_3)P(C_R)P(R_I) - P(A_C)P(C_1)P(C_R)P(R_I) +$
$P(A_C)P(C_1)P(C_3)P(C_R)P(R_I) - P(C_1)P(C_R)P(C_R)P(R_I) +$
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_R)P(C_R)P(R_I) -$
$P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) +$
$P(C_2)P(C_R)P(R_I) - P(C_1)P(C_2)P(C_R)P(R_I) -$
$P(A_C)P(C_2)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_2)P(C_R)P(R_I) -$
$P(C_2)P(C_R)P(C_R)P(R_I) + P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(C_2)P(C_R)P(C_R)P(R_I) - P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$
$P(C_R)P(R_I)P(R_I) - P(C_1)P(C_R)P(R_I)P(R_I) -$
$P(C_3)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_3)P(C_R)P(R_I)P(R_I) -$
$P(C_R)P(C_R)P(R_I)P(R_I) + P(C_1)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) - P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) - P(C_1)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) -$
$P(C_2)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) +$
$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) -$
$P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) +$
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) +$
$P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) -$
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)P(R_I) +$
$P(C_1)P(C_R)P(C_R)P(R_I)P(R_I) - P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(A_C)P(C_1)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) - P(A_C)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I)$
$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) +$

$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) - P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) -$
$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) -$
$P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$
$P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)]$.

By cancellation,

$$\frac{G_1G_2}{1 - G_1G_2H} = 1 - [1 + P(C_1)P(C_2)P(C_R)P(R_I) + P(C_2)P(C_R)P(R_I)P(R_I) -$$
$P(C_1)P(C_2)P(C_R)P(R_I)P(R_I) - P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(C_1)P(C_2)P(C_R)P(C_R)P(R_I) + P(C_1)P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(C_2)P(C_R)P(C_R)P(R_I)P(R_I) - P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) +$
$2 \times P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) -$
$P(A_C)P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) +$
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) - 2 \times P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I) +$
$2 \times P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$
$2 \times P(A_C)P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$
$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) +$
$P(A_C)P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I) -$
$P(A_C)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) - P(A_C)P(C_1) -$
$P(C_1)P(C_3)P(C_R)P(R_I) + P(A_C)P(C_1)P(C_3)P(C_R)P(R_I) -$
$P(A_C)P(C_2)P(C_R)P(R_I) - P(C_3)P(C_R)P(R_I)P(R_I) +$
$P(C_1)P(C_3)P(C_R)P(R_I)P(R_I) + P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(A_C)P(C_1)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) +$
$P(A_C)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I) -$
$P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R)P(R_I)P(R_I)]$.

By combining similar terms,

$$\frac{G_1G_2}{1 - G_1G_2H} = P(A_C)P(C_1) + [-P(C_1)P(C_2) + P(C_1)P(C_3) + P(A_C)P(C_2) - P(A_C)P(C_1)P(C_3)]P(C_R)P(R_I) + [-P(C_2) + P(C_3)][1 - P(C_1)]P(C_R)P(R_I)P(R_I) + [1 - P(A_C)][1 - P(A_C)]P(A_C)P(C_1)P(C_2)P(C_3)P(C_R)P(C_R) + [1 - P(A_C)] \times \{P(C_1)[P(C_2) - P(C_3)] + P(A_C)P(C_3)[P(C_1) + P(C_2) - 2P(C_1)P(C_2)]\}P(C_R)P(C_R)P(R_I) + [1 - P(C_1)] \times \{P(C_2) - P(C_3) + P(A_C)P(C_3)[1 - P(C_2)]\} \times P(C_R)P(C_R)P(R_I)P(R_I).$$

By Scott's definitions of the variables, it would be expected that:

a. The recovery block, G_1, with its n alternates be a function of $P(C_i)$;

b. The acceptance test, G_2, be a function of $P(A_C)$ and/or $P(R_I)$; and

c. The rollback, H, be a function of $P(C_R)$.

Although this was true for the special case with n = 1, the cases with n = 2 and n = 3 show that the two models are too distinct in their methods to be compared.

APPENDIX VI

ANALYSIS OF SCOTT'S N-VERSION PROGRAMMING RELIABILITY MODEL

In Scott's N-version programming reliability model, the variables are defined as:

$P(C_i)$ = the probability of version i executing correctly;

$P(I_i)$ = the probability of version i executing incorrectly, $1 - P(C_i)$;

Type 1 Error = all outputs disagree;

Type 2 Error = an incorrect output occurs more than once;

Type 3 Error = there is an error in the voting (decision algorithm) procedure;

n = the number of versions; and

R = the reliability.

The results of Scott's N-version programming reliability model will be compared to those that would be obtained in the software reliability model that has been proposed. The block diagram for the N-version software that is described by Scott would look like:

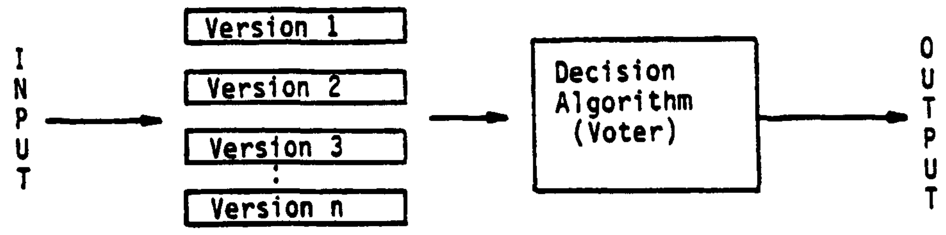


Figure 18. Basic N-Version Software

For the proposed software reliability model, it is desired that the individual reliability values for the n versions be combined to form one reliability value (or transfer function) for the group. The block diagram for this might be

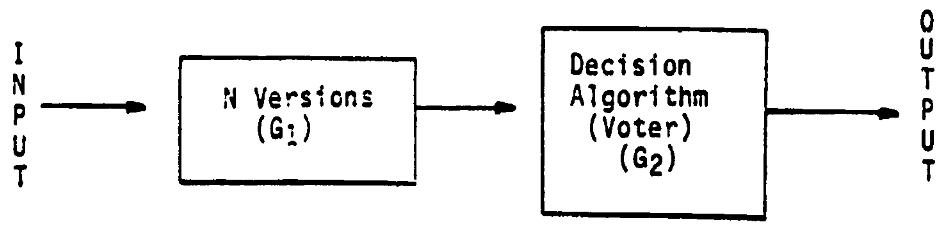


Figure 19. Basic N-Version Software Equivalent

with an overall transfer function of $R = G_1 \times G_2$.

A simple case to examine is the N-version software composed of three independent versions. The probability of a system error in this 3-version software system becomes the probability of at least two versions executing incorrectly. This simple case is computed with Scott's N-version programming reliability model as

$$R = 1 - [\text{Type 1 Error} + \text{Type 2 Error} + \text{Type 3 Error}].$$

In Scott's analysis, he assumes that the probability of a Type 3 Error is zero. Therefore, with Scott's model,

$$R = 1 - \{[P(I_1)P(I_2)P(I_3)] + [P(C_1)P(I_2)P(I_3) + P(I_1)P(C_2)P(I_3) + P(I_1)P(I_2)P(C_3)] + 0\}.$$

By substitution,

$$\begin{aligned} G_1G_2 = 1 - \{&[1 - P(C_1)][1 - P(C_2)][1 - P(C_3)] + \\ &P(C_1)[1 - P(C_2)][1 - P(C_3)] + \\ &[1 - P(C_1]P(C_2)[1 - P(C_3)] + \\ &[1 - P(C_1)][1 - P(C_2)]P(C_3)\}. \end{aligned}$$

Multiplying out the factors gives

$$\begin{aligned} G_1G_2 = 1 - \{&[1 - P(C_1) - P(C_2) + P(C_1)P(C_2)][1 - P(C_3)] + \\ &P(C_1)[1 - P(C_2) - P(C_3) + P(C_2)P(C_3)] + \\ &P(C_2)[1 - P(C_1) - P(C_3) + P(C_1)P(C_3)] + \\ &P(C_3)[1 - P(C_1) - P(C_2) + P(C_1)P(C_2)]\}. \end{aligned}$$

With further multiplication,

$$\begin{aligned}G_1G_2 = 1 - [&1 - P(C_1) - P(C_2) + P(C_1)P(C_2) - P(C_3) + \\ &P(C_1)P(C_3) + P(C_2)P(C_3) - P(C_1)P(C_2)P(C_3) + \\ &P(C_1) - P(C_1)P(C_2) - P(C_1)P(C_3) + \\ &P(C_1)P(C_2)P(C_3) + P(C_2) - P(C_1)P(C_2) - \\ &P(C_2)P(C_3) + P(C_1)P(C_2)P(C_3) + P(C_3) - \\ &P(C_1)P(C_3) - P(C_2)P(C_3) + P(C_1)P(C_2)P(C_3)].\end{aligned}$$

Cancelling alike terms gives

$$\begin{aligned}G_1G_2 = 1 - [&1 - P(C_1)P(C_2) - P(C_1)P(C_3) - P(C_2)P(C_3) + \\ &2P(C_1)P(C_2)P(C_3)].\end{aligned}$$

This can be reduced to

$$G_1G_2 = P(C_1)P(C_2) + P(C_1)P(C_3) + P(C_2)P(C_3) - 2P(C_1)P(C_2)P(C_3).$$

With Scott's assumption that the probability of a Type 3 Error is zero, the equivalent assumption in the software reliability model would be that $G_2 = 1$. Therefore,

$$G_1 = P(C_1)P(C_2) + P(C_1)P(C_3) + P(C_2)P(C_3) - 2P(C_1)P(C_2)P(C_3).$$

TECHNICAL REPORT

on

SOFTWARE RELIABILITY DATA AND DATABASE

1.0 INTRODUCTION

The software reliability data required by the software reliability model was described in the previous section. This section describes the data to be collected by avionics systems developers prior to starting development of the software packages, during the development of the software packages, and during the operational life of the software. In addition, attributes of the data base program are described.

2.0 BACKGROUND

As noted in the previous section of this report, the software reliability model's primary inputs are probabilities. While this is an excellent form for the model, it is not the normal type of data collected by avionics systems developers. The data collected by developers of avioncs systems tends to be deterministic as opposed to statistical. The statistical and probabilistic values are derived from the determinstic data as discussed in earlier sections of this report. The method used by Battelle to derive these data ties in closely with the database program.

Although any database manager can store and retrieve information, spreadsheet programs provide more capability than a simple file manager. The spreadsheet programs with their built-in functions provide the capability to analyze the data instead of simply storing and retrieving data. There are a large number of spreadsheet programs to choose from and the selection of a specific program is not within the scope of this work. This section does discuss factors which should be considered in the selection of a spreadsheet program.

3.0 DATABASE PROGRAM AND SOFTWARE RELIABILITY DATA

The data to be collected by the avionics system developer is collected at different times during the software life cycle. It is likely to be organized in any database program as an assortment of different files. In order to manipulate these data contained in different files into the form required by the software reliability model, a relational database manager which can link separate data files to create a database containing information selected from the different files is recommended.

Principles of data design should be applied when developing the database. Data design is a set of principles and analytic tools that brings to the design of data the same kind of organization that structured programs brings to programs. To avoid dangerous file designs, a clear understanding of the dependencies in the files is necessary. Transitive dependencies are a frequent cause of structural problems in the design of files.

Once the database is created, use of a spreadsheet with its built-in arithmetic and statistical functions will provide the capability to analyze deterministic data such as lines of source code, memory usage, errors detected, errors corrected, and linkages and assemble these data into the statistical form required for determining the probabilistic inputs required by the software reliability model.

The inputs required for the database program will depend upon the software reliability model (reference Section 2, "Technical Report on Review of Previous Studies of Software Reliability Models" for some examples) used to obtain the probabilistic reliability value that is required in the software reliability model described in Section 4, "Technical Report on Formulation of the Software Reliability Model". Table 1 lists some of the software reliability models and their required inputs. The built-in arithmetic and statistical functions will manipulate this input data to obtain the probabilistic reliability values. An example of what the input and output for the database program might look like is given in Tables 2 and 3.

Table 1. Input Data Used by Various Software Reliability Models

Software Reliability Model	Input Data
Generalized Imperfect Debugging Model	N = the total number of errors; p = the probability of perfect programmer debugging behavior
Bug-Proportional Model	$\epsilon_r(\tau)$ = the number of remaining bugs
Geometric Poisson Model	λ = the average number of faults occurring in the first interval; t_i = the i-th debugging interval
Schneidewind Non-Homogeneous Poisson Model	m_i = the estimated number of errors in interval i
Jelinski-Moranda De-Eutrophication Model	N = the number of initial errors in the program; X_i = the length of the i-th debugging interval; n = the number of errors found to date
Extended Jelinski-Moranda Model	N = the total number of initial errors; n_i = the cumulative number of errors found through the i-th interval; t_i = the i-th debugging interval
Geometric De-Eutrophication Model	D = the initial error detection rate; n = the total number of errors discovered; X_i = the i-th debugging interval
Lines of Source Code Model	n = the total number of lines of source code; the programming language (i.e., FORTRAN, Cobol, Ada, etc.)

Errors Detected	Time Interval	Cumulative Errors Removed	Cumulative Time	Cumulative Errors Detected
2	7	0	7	2
4	21	1	28	6
0	8	4	36	6
0	2	5	38	6
3	3	6	41	9
2	4	8	45	11
11	6	12	51	22
3	1	13	52	25
0	2	23	54	25
1	1	24	55	26
1	3	26	58	27
1	2	27	60	28
1	2	28	62	29
0	3	29	65	29
1	6	30	71	30
1	23	31	94	31
2	4	32	98	33
1	9	34	107	34

Table 2. Possible Input to the Database Program [ANGUS]

Model	Estimated N	Observed Errors	Estimated ϕ
Geometric Poisson	75	34	0.0056
Non-Homogeneous Poisson	75	34	0.0056
Geometric Poisson	16	15	0.1586
Non-Homogeneous Poisson	16	15	0.1727
Geometric Poisson	49	20	0.0343
Non-Homogeneous Poisson	49	20	0.0349
Generalized Poisson	21	20	0.1338
Generalized Poisson	28	23	0.2072
IBM Poisson (Modified)	37	23	0.0082
Geometric Poisson	155	73	0.0204
Non-Homogeneous Poisson	155	73	0.0206

Table 3. Possible Input and Output for the Database Program [ANGUS]

4.0 REFERENCES

[ANGUS] Angus, J.E., Bowen, J.B., and VanDenBerg, S.J., Reliability Model Demonstration Study, Hughes Aircraft Company, RADC-TR-83-207, Volume I, August 1983, pp. 4-2 and 4-26 through 4-31.

[WEISS] Weiss, David M., "A Comparison of Errors in Different Software-Development Environments", Naval Research Laboratory, Washington, D.C., Report Number AD-A118-296, 14 July 1982.